

Inhaltsverzeichnis

1 EINLEITUNG	6
1.1 Projektbeschreibung.....	6
1.2 Rahmenbedingungen.....	7
2 PROJEKTAUFBAU.....	10
2.1 Vorüberlegungen	10
2.2 Blockschaltbild.....	12
2.3 Zeitplan	13
3 ALLGEMEINES.....	14
3.1 Hardware	14
3.1.1 Zilog Z80	14
3.1.2 PIO (Parallel Input/Output)	14
3.1.3 SIO (Seriell Input/Output) – RS232	15
3.1.4 Pull-Up-Widerstände	15
3.1.5 Pull-Down-Widerstände	15
3.1.6 RAM (Random Access Memory).....	16
3.1.7 ROM (Read Only Memory).....	16
3.2 Software	17
3.2.1 Assemblersprache	17
3.2.2 Programmiersprache ‚C‘	17
3.2.3 Compiler	18
3.2.4 Linker	18
3.2.4.1 Statisches Linken	18
3.2.4.2 Dynamisches Linken	18
3.2.5 Interrupt	19
3.2.6 Begrifflichkeiten der Informatik.....	20
4 HARDWARE	21
4.1 Z80–EMUF–Platine.....	21
4.1.1 Konstruktionsüberlegungen	21
4.1.2 Aufbaubeschreibung	21
4.1.2.1 CPU (Central-Prozessing-Unit)	22
4.1.2.2 ALU (Arithmetic Logic Unit)	22
4.1.2.3 CU (Control Unit).....	22
4.1.2.4 Interner Speicher (Akku)	23
4.1.2.5 Ablauf einer Befehlsausführung.....	23
4.1.2.6 Systembus.....	24
4.1.2.6.1 Datenbus	24
4.1.2.6.2 Adressbus.....	24
4.1.2.6.3 Steuerbus.....	25
4.1.3 Zusätzliche Änderungen.....	26
4.1.3.1 Jumperbelegungen für ROM und RAM	27
4.1.4 Schaltplan Z80-EMUF-Platine	28
4.1.5 Bestückungsplan Z80-EMUF-Platine.....	30

4.1.6 Adressdecoder	31
4.1.6.1 Blockschaltbild Adressdecoder	32
4.1.6.2 Steueradressen	33
4.1.7 SIO	34
4.2 PIO-Verteiler-Platine	37
4.2.1 Konstruktionsüberlegungen	37
4.2.2 Aufbaubeschreibung	39
4.2.2.1 PIO-Controller	40
4.2.2.2 Daisy-Chain	42
4.2.2.3 Adressierung der PIO-Register	42
4.2.2.4 Initialisierung des PIO-Controllers	43
4.2.3 Schaltplan PIO-Verteiler-Platine	44
4.2.4 Layout PIO-Verteiler-Platine (Oberseite)	45
4.2.5 Layout PIO-Verteiler-Platine (Unterseite)	45
4.2.6 Bestückungsplan PIO-Verteiler-Platine	46
4.3 CPU-Extension-Platine	47
4.3.1 Konstruktionsüberlegungen	47
4.3.2 Aufbaubeschreibung	47
4.3.3 Schaltplan CPU-Extension	49
4.3.4 Layout CPU-Extension (Oberseite)	50
4.3.5 Layout CPU-Extension (Unterseite)	50
4.3.6 Bestückungsplan CPU-Extension	50
4.4 PID-IN-Platine	51
4.4.1 Aufbaubeschreibung	51
4.4.2 Schaltplan PID-IN-Platine	51
4.4.3 Layout PID-IN-Platine	52
4.4.4 Bestückungsplan PID-IN-Platine	52
4.5 PID-Sticks (Personen-ID-Sticks)	53
4.5.1 Konstruktionsüberlegungen	53
4.5.2 Aufbaubeschreibung	53
4.5.3 Schaltplan PID-Sticks	54
4.5.4 Layout PID-Sticks	55
4.5.5 Bestückungsplan PID-Sticks	55
4.6 DISTAS (Display- und Taster-Platine)	56
4.6.1 Konstruktionsüberlegungen	56
4.6.2 Aufbaubeschreibung	57
4.6.3 Schaltplan DISTAS	58
4.6.4 Layout DISTAS (Oberseite)	59
4.6.5 Layout DISTAS (Unterseite)	59
4.6.6 Bestückungsplan DISTAS	60
4.7 Gehäuse	61
4.7.1 Konstruktionsüberlegungen	61
4.7.2 Aufbaubeschreibung	61
4.7.3 Frontplatte	62
5 SOFTWARE	63
5.1 Softwarestruktur	63
5.1.1 Visuelle Struktur der Software	64
5.2 Headerdatei „io.h“	66

5.3 Systemmodule	66
5.3.1 Systemmodul ,delay()'	66
5.3.2 Systemmodul ,busy1()'	66
5.3.3 Systemmodul ,output()'	67
5.3.4 Systemmodul ,in()'	67
5.3.5 Systemmodul ,put1()'	67
5.3.6 Systemmodul ,outstr1()'	67
5.3.7 Systemmodul ,lcd1()'	67
5.4 Startmodul ,akemain()'	68
5.5 Hauptprogramm ,main()'	68
5.5.1 Programmbeschreibung	68
5.5.2 Programmablaufplan ,main()'	69
5.6 Interruptroutinen.....	74
5.6.1 Programmbeschreibung irouta() und iroutb().....	74
5.6.2 Programmablaufplan ,irouta()'	75
5.6.3 Programmablaufplan ,iroutb()'	76
5.7 Unterprogramme.....	77
5.7.1 init().....	77
5.7.1.1 Programmbeschreibung	77
5.7.1.2 Programmablaufplan ,init()'	78
5.7.2 iniim2()	79
5.7.2.1 Programmbeschreibung	79
5.7.2.2 Programmablaufplan ,iniim2()'	80
5.7.3 typ1()	81
5.7.3.1 Programmbeschreibung	81
5.7.3.2 Programmablaufplan ,typ1()'	82
5.7.4 pid().....	84
5.7.4.1 Programmbeschreibung	84
5.7.4.2 Programmablaufplan ,pid()'	84
5.7.5 pressu().....	86
5.7.5.1 Programmbeschreibung	86
5.7.5.2 Programmablaufplan ,pressu()'	86
5.7.6 start()	88
5.7.6.1 Programmbeschreibung	88
5.7.6.2 Programmablaufplan ,start()'	88
5.7.7 count11().....	89
5.7.7.1 Programmbeschreibung	89
5.7.7.2 Programmablaufplan ,count11()'	89
5.7.8 ziel()	91
5.7.8.1 Programmbeschreibung	91
5.7.8.2 Programmablaufplan ,ziel()'	92
5.7.9 count12().....	95
5.7.9.1 Programmbeschreibung	95
5.7.9.2 Programmablaufplan ,count12()'	95
5.7.10 count13().....	96
5.7.10.1 Programmbeschreibung	96
5.7.10.2 Programmablaufplan ,count13()'	98
5.7.11 ende1()	101
5.7.11.1 Programmbeschreibung	101
5.7.11.2 Programmablaufplan ,ende1()'	102
5.7.12 art1()	104
5.7.12.1 Programmbeschreibung	104
5.7.12.2 Programmablaufplan ,art()'	104
5.7.13 auswahl()	105
5.7.13.1 Programmbeschreibung	105

5.7.14 led11()	106
5.7.14.1 Programmbeschreibung	106
5.7.14.2 Programmablaufplan ,led11()'	106
5.7.15 led1aus()	107
5.7.15.1 Programmbeschreibung	107
5.7.15.2 Programmablaufplan ,led1aus()'	107
5.7.16 blink1()	108
5.7.16.1 Programmbeschreibung	108
5.7.16.2 Programmablaufplan ,blink1()'	109
5.8 Datenübermittlung zwischen System und PC	111
6 INBETRIEBNAHME	112
6.1 Testphase	112
6.1.1 Testprotokoll des ersten Testprogramms ,check1()'	113
6.2 Aufgetretene Probleme und Lösungen	114
8 PROJEKTTAGEBUCH	118
9 BEDIENUNGSANLEITUNG TGK3000	120
9.1 Allgemeines	120
9.2 Aufbau der der TGK3000	120
9.3 Menüstruktur der TGK3000	120
9.3 Menüstruktur der TGK3000	121
9.4 Auswahlmöglichkeiten der einzelnen Abfragen	124
10 TABELLEN	125
10.1 Signallisten	125
10.2 Steckerbelegungen	127
10.3 Stücklisten	130
10.3.1 Z80-EMUF	130
10.3.2 DISTAS	131
10.3.3 PIO-Verteiler-Platine	131
10.3.4 CPU-Extension-Platine	132
10.3.5 PID-IN-Platine	132
10.3.6 PID-Stick	132
10.4 Kostenaufstellung	132
11 ERKLÄRUNG DER PROJEKTTEILNEHMER	133
12 QUELLENNACHWEISE	134

13 ANHANG	135
13.1 Quellcode	135
13.2 Datenblätter	136

1 Einleitung

1.1 Projektbeschreibung

Dieses Projekt wurde im Rahmen der Weiterbildung zum staatlich geprüften Techniker für Elektrotechnik innerhalb einer zehnwöchigen Projektphase erstellt. Die Projektgruppe setzte sich aus den Schülern Eike Grünhagen, Esmail Kiapur sowie Carsten Töben zusammen.

Bei unseren ersten Vorüberlegungen stand schnell fest, dass es sich nicht um ein von der Schule vorgegebenes Projekt handeln sollte. Wir hatten das Bestreben, etwas mit praktischem Bezug und späterem Nutzwert zu erarbeiten.

Wir einigten uns darauf, eine automatische und digitalisierte Version von bereits bestehenden Atemschutzgeräteüberwachungen zu entwickeln. Dies sind Geräte, welche im Alltag von Feuerwehren der Überwachung der eigenen Kameraden im Atemschutzeinsatz dienen.

Die schon existierenden Geräte von namhaften Herstellern sind meistens analoger Natur oder können aus finanziellen Gründen für kleinere Feuerwehren nicht beschafft werden.

Ein anderer Hintergrund war für uns jedoch das überzeugende Argument. Bei den Feuerwehren steigt, wie in vielen anderen Bereichen auch, der Verwaltungsaufwand immer mehr an. Dies steht schwindenden Mitgliederzahlen entgegen.

Somit sollte unser System sämtliche Einsatzdaten, welche ohnehin im Nachhinein dokumentiert und festgehalten werden müssen, speichern können. Diese gespeicherten Daten, sowie evtl. zusätzlich resultierende Daten, sollen im Anschluss an den Einsatz mittels serieller Datenschnittstelle an einen PC übertragen werden können.

Eine weitere Herausforderung war es, dieses Gerät mit einer simplen Menüsteuerung sowie wenigen Tasten bedienbar zu machen. Dies ergab sich aus der Tatsache, dass bei Einsätzen mit Atemschutz immer zusätzlicher Stress entsteht. Somit hat der Bediener des Überwachungsgerätes nicht die Möglichkeit und vor allem nicht die Zeit, sich durch sämtliche Menüs „kämpfen“ zu müssen, um eine korrekte Atemschutzüberwachung zu gewährleisten.

Es ergab sich eine ehrgeizige Herausforderung, der wir uns gerne stellten.

1.2 Rahmenbedingungen

Zum besseren Verständnis einer Einsatzsituation, in der ein solches Überwachungsgerät verwendet wird, bedarf es für Personen, die nichts oder wenig mit dem Feuerlöschwesen zu tun haben, einer kurzen Erläuterung.

Atemschutzgeräte, in den unterschiedlichsten Ausführungen, werden immer dann eingesetzt, wenn in der Umgebungsluft, der sich Feuerwehrleute aussetzen müssen, Atemgifte vorhanden sind. Außerdem kann es vorkommen, dass nicht genügend Sauerstoff in der Atemluft enthalten ist. Auch dies erfordert den Einsatz von Atemschutzgeräten.

In der Vergangenheit kam es leider immer wieder zu Unfällen mit schlimmen Folgen, so dass an dieser Stelle geeignete Ausbildungs- und Überwachungsverfahren eingeführt werden mussten. Unser System soll zu einer besseren und effizienteren Kontrolle der vorgehenden Trupps beitragen.



Rüstet sich ein Trupp, der sich je nach Atemschutzgeräteart aus zwei oder drei Feuerwehrleuten zusammensetzt, aufgrund der o.g. Umstände mit Atemschutz aus, muss daraus resultierend automatisch entschieden werden, dass eine Überwachung des Trupps stattfinden muss. Diesen Auftrag erhält ein ausgebildeter Feuerwehrangehöriger (FA), der anschließend während des gesamten Einsatzgeschehens dafür zuständig ist.

Ein Einsatz unter bisherigen Umständen läuft wie folgt ab.

Der Überwachende bereitet die Datenaufnahme vor, indem er von den vorgehenden Truppmitgliedern die Personenplaketten mit dem sich daran befindlichen Karabinerhaken in die „Überwachungstafel“ einhängt. Die FA geben ihre Behälterdrücke, welche von an den Geräten montierten Manometern abgelesen werden können, an. Diese werden in Verbindung mit den Namen notiert.



Beginnt nun der Einsatz, zwischen Vorbereitung und Einsatzbeginn können durchaus mehrere Minuten liegen, wird dies dem Überwachenden mitgeteilt. Dieser zieht nun eine, am Gerät angebrachte, handelsübliche Eieruhr auf eine vordefinierte, maximale Einsatzzeit auf. Diese Zeit richtet sich nach der eingesetzten Geräteart.

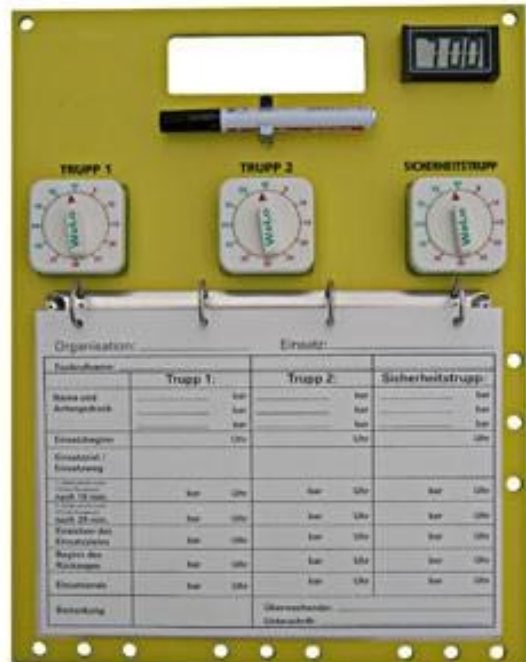
Der Trupp macht sich auf den Weg zur unmittelbaren Einsatzstelle. Dies kann in einigen Fällen, z.B. bei einem Zimmerbrand in einem verrauchten Mehrfamilienhaus, durchaus einige Minuten in Anspruch nehmen.

Dort angekommen, meldet er sich als erstes mit einem Handsprechfunkgerät bei der Atemschutzgeräteüberwachung. Der Truppführer gibt die einzelnen Behälterdrücke durch und der Trupp beginnt mit seiner Arbeit. Der Überwachende notiert diese und ermittelt nun anhand mehrerer Tabellen (abhängig vom Startdruck der Truppmitglieder) die am Einsatzort maximal zu verbleibende Arbeitszeit. Dieser Schritt erfordert viel Konzentration und wird in der Realität leider nicht immer durchgeführt.

Die Eieruhr wird nun auf die neu ermittelte Restzeit gestellt. Spätestens wenn diese abgelaufen ist, muss der Trupp vom Überwachenden über Funk aufgefordert werden, unverzüglich den Rückzug anzutreten.

Ist der Trupp außerhalb der Einsatzstelle, meldet er sich umgehend bei der Atemschutzüberwachung zurück und gibt den Druck an, der sich nun noch in den Atemluftflaschen befindet. Auch diese Werte werden vom zuständigen FA aufgenommen. Außerdem notiert dieser nun die gesamte Einsatzzeit und die verrichtete Tätigkeit des Trupps.

Im Anschluss an den Einsatz müssen die gesamten notierten Daten nun wiederum von der „Überwachungstafel“ in so genannte Dokumentationshefte übertragen und ggf. ausgewertet werden.



Unsere Absicht ist es nicht, dieses Vorgehen stark zu verändern, sondern lediglich ein wenig zu erleichtern.

Jeder FA erhält im Vorfeld für seine gesamte Dienstzeit einen eigenen Personen-Identitäts-Stick, auf dem ein persönlicher Code gespeichert ist.

Rüstet sich nun ein Trupp aus, geben die Truppmitglieder ihren ID-Stick an der Überwachung ab. Der zuständige FA wählt die zu verwendende Gerätevariante und

steckt anschließend die PID-Sticks in die dafür vorgesehenen Buchsen. Er wird nun vom System aufgefordert, die Behälterdrücke in das System einzugeben.

Beginnt der Trupp seinen Einsatz, erfolgt eine Start-Bestätigung im System. Automatisch beginnt die, anhand der Gerätevariante, vom System ermittelte maximale Einsatzzeit herunter zu zählen.

Kommt der Trupp an der Einsatzstelle an, meldet dieser sich ebenfalls über ein Funkgerät bei der Überwachung. Die Behälterdrücke werden durchgesagt und erneut in das System eingegeben. Automatisch wird nun die noch verbleibende Arbeitszeit ermittelt und erneut auf dem Display herunter gezählt.

Ist diese Zeit abgelaufen, wird der Überwachende durch ein akustisches Signal aufgefordert, den Trupp zurückzuordern.

Dieses wird am Gerät bestätigt und von diesem Zeitpunkt an wird die tatsächlich benötigte Zeit für den Rückweg angezeigt.

Ist der Einsatz beendet, gibt der Überwachende den Restdruck der Behälter ein und auch die verrichtete Tätigkeit wird digital aufgenommen.

Im Feuerwehrhaus angekommen ist nun lediglich das System an einen bereits vorhandenen PC anzuschließen und alle relevanten Einsatzdaten werden automatisch an eine Datenbank übermittelt.

Aus dieser Erläuterung soll hervorgehen, dass unsere Entwicklung einige Schritte im Einsatzgeschehen erleichtern wird. Auf die Verwendung von Funkgeräten sowie bereitstellen von geeignetem Überwachungspersonal kann nach wie vor nicht verzichtet werden.

Im Übrigen steht die Entwicklung der soeben angesprochenen Datenbank, welche mit Hilfe der Programmiersprache JAVA realisiert werden soll, bei unserer Projektarbeit zunächst im Hintergrund, bis die gesamte Hardware einwandfrei funktioniert.

2 Projektaufbau

2.1 Vorüberlegungen

Für die Realisierung des Projektes stand zunächst die Entscheidung aus, welches Mikroprozessor-System (μ P) wir verwenden sollten, um möglichst schnell und kostengünstig ans Ziel zu gelangen. Da es eine Vielzahl von μ P-Systemen gibt, die wiederum mit einer Vielzahl von Funktionen und großem Befehlsumfang aufwarten, entschieden wir uns für ein vertrautes μ P-System, dem Zilog Z80, bei dem wir den Befehlsumfang bereits kennen und dessen Funktionen und Leistung für diese Aufgabe voll und ganz ausreichend sind. Da wir aufgrund der Bedingungen für den praktischen Einsatz entsprechend mehr Hardware benötigen (3 Trupps, 3 LCD, 9 ID-Sticks, ...), mussten wir entweder ein komplett neues EMUF-System mit zusätzlicher Peripherie „onboard“ entwickeln, oder ein vorhandenes Standard-EMUF-System verwenden und dazu die fehlende Peripherie entwickeln.

Um das Rad nicht neu erfinden zu müssen, entschieden wir uns eine bewährte Standard-EMUF-Platine der Firma „Kanis“ zu verwenden und die fehlende Peripherie auf Zusatzplatinen selbst zu entwickeln und zu fertigen. Somit haben wir bereits im Voraus eventuelle Fehlerquellen bei der Entwicklung eines neuen μ P-EMUF-Systems vermieden.

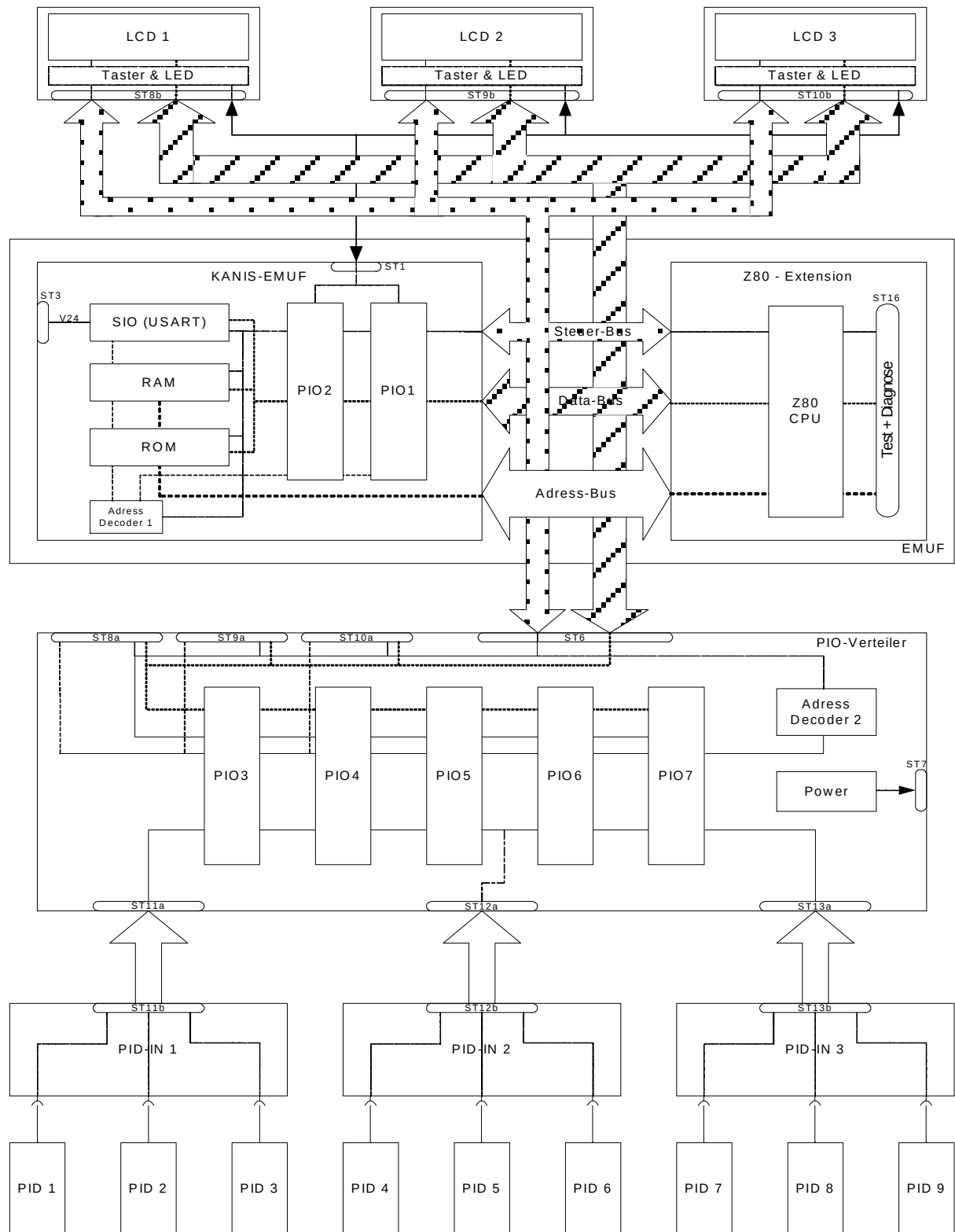
Zum Einlesen der PID-Sticks gab es viele verschiedene Möglichkeiten. Um uns einen erheblichen Hardwareaufwand zu ersparen, wollten wir einen analogen PID-Stick entwickeln, der die Kodierung der ID`s durch festgelegte Widerstandswerte vornimmt. Dazu benötigten wir jedoch einen analogen Eingang mit AD-Wandler pro PID-Stick, oder man würde nur einen analogen Eingang mit nur einem AD-Wandler verwenden, der dann gemultiplext würde.

Eine weitere Überlegung war, die PID-Sticks digital zu codieren. Dazu wären jedoch entsprechend der Anzahl der PID-Sticks mehrere Datenleitungen pro PID-Stick nötig, die dann über einen parallelen Port einer PIO einzulesen wären. Dann stellt sich wiederum die Frage, ob man dafür nur eine PIO benutzt, die wiederum auf neunfach gemultiplext wird, oder man würde pro PID-Stick einen kompletten Port einer PIO verwenden und das μ P-System um entsprechend viele PIO`s ergänzen. Um der Problematik des analogen Einlesens und externen Multiplexens zu entgehen, entschieden wir uns für den komplett parallelen Aufbau der PID-Sticks über eine digitale Kodierung und somit für die Erweiterung des Systems um fünf PIO`s.

Für den praktischen Einsatz des Gerätes ist es notwendig, eine spannungsversorgungsfreie Betriebslaufzeit von mind. zwei Stunden zu erreichen. Dementsprechend benötigen wir einen Akku mit entsprechender Kapazität. Um den Ladezustand während des Betriebes zu überwachen soll eine Akkuzustandsanzeige mit Bargraphanzeige entwickelt werden.

Zusätzlich fehlt ein integrierter Laderegler, der den Akku nach dem Einsatz kontrolliert wieder auflädt.

2.2 Blockschaltbild



2.3 Zeitplan

ID	Task Name	Start	Finish	Duration	Feb 2006			Mar 2006			Apr 2006			May 2006			Jun 2006							
1	Planung Hardware	06.02.2006	10.02.2006	5d	5.2	12.2	19.2	26.2	5.3	12.3	19.3	26.3	2.4	9.4	16.4	23.4	30.4	7.5	14.5	21.5	28.5	4.6	11.6	18.6
2	Planung Kaskadierung Adressdecoder für 11 Peripheriebausteine	13.02.2006	15.02.2006	3d																				
3	Layoutentwicklung (PIO-Verteiler)	15.02.2006	28.02.2006	10d																				
4	Softwareentwicklung Hauptroutinen	15.02.2006	28.02.2006	10d																				
5	Layoutentwicklung Display	01.03.2006	06.03.2006	4d																				
6	Layoutentwicklung PID / PID-IN	01.03.2006	06.03.2006	4d																				
7	Platinenerstellung (ätzen / bestücken)	06.03.2006	17.03.2006	10d																				
8	Software Hauptroutinen auf EMES Testen	06.03.2006	13.03.2006	6d																				
9	Softwareentwicklung Testprogramm für Inbetriebnahme PIO-Verteiler	14.03.2006	15.03.2006	2d																				
10	Inbetriebnahme Hardware mit Testprogramm (Kans-EMUJ, PIO-Verteiler, Displays, PID-IN)	15.03.2006	24.03.2006	8d																				
11	Testphase mit Test- / Hauptprogramm	24.03.2006	07.04.2006	11d																				
12	Fehlersuche Hardware	10.04.2006	17.04.2006	6d																				
13	Fehlersuche Software	12.04.2006	20.04.2006	7d																				
14	Fehlerbehebung (Hardwareänderungen)	17.04.2006	24.04.2006	6d																				
15	Gehäuseplanung	24.04.2006	28.04.2006	5d																				
16	Dokumentation	17.04.2006	05.05.2006	15d																				
17	Reserve	08.05.2006	12.05.2006	5d																				
18	Deadline	15.05.2006	15.05.2006	1d																				

3 Allgemeines

3.1 Hardware

3.1.1 Zilog Z80

Der Z-80 zählt zu den erfolgreichsten Prozessoren aller Zeiten. Der Hersteller Zilog nahm sich Intels 8080 zum Vorbild und verbesserte ihn sowohl von der Integrierbarkeit in Rechnersysteme als auch vom Maschinenbefehlssatz her. Dadurch entstand ein sehr leistungsfähiger Prozessor, der allerdings wegen seiner ausgesprochen komplexen Befehle auch mit deutlich höherem Takt als sein spartanischerer Hauptkonkurrent 6502 betrieben werden musste, um die gleiche Geschwindigkeit zu entwickeln. Die typische Taktfrequenz lag bei 3,5 - 4 MHz. Der wichtigste Meilenstein bei der Verbreitung des Z-80 war wohl das Betriebssystem CP/M von Digital Research. Ähnlich wie heute Windows 95/98 nur auf Intel-CPU's und Kompatiblen läuft, war CP/M lange Zeit nur für den Z-80 zu bekommen. Dadurch entstand eine monopolähnliche Position für den Z-80 bei Bürocomputern der späten 70er und frühen 80er Jahre. Später kamen auch etliche Homecomputer dazu, angefangen vom Sinclair ZX-80, ZX-81 und Spektrum, den Amstrad-Computern bis hin zu den Rechnern des in Europa wenig erfolgreichen MSX-Standards.

3.1.2 PIO (Parallel Input/Output)

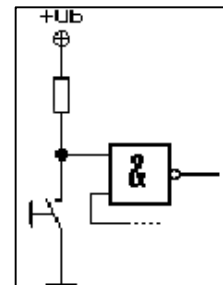
Der Z80-PIO-Controller ist eine frei programmierbare parallele Ein- / Ausgabe-Schnittstelle zwischen System und Peripherie, welche bei der Z80 Familie meistens als externer Baustein ausgeführt ist. Er hat zwei Kanäle (Ports) zu je 8 Bit, die wahlfrei (auch bitweise) und unabhängig voneinander angesteuert und programmiert werden können. Es ist möglich mehrere PIO-Bausteine via Adress-Logik zu kaskadieren, maximal jedoch 128 Bausteine im Gesamtsystem. Jeder Kanal verfügt über eine separate Interrupt-Logik, wobei Kanal A jedoch grundsätzlich die höhere Priorität besitzt. Die Interrupt-Prioritäten bei mehreren PIO-Controllern werden mit Hilfe der so genannten Daisy-Chain geregelt, eine art Ablaufkette die alle nachfolgenden Komponenten in der Kette sperrt, wenn eine Komponente einen Interrupt auslöst. (siehe Kapitel Hardware-PIO)

3.1.3 SIO (Seriell Input/Output) – RS232

Die meistverbreitete serielle Verbindung zwischen Computern und deren Peripherie ist die RS232-Schnittstelle, auch bekannt unter der Bezeichnung V.24-Schnittstelle. Als Steckverbindung wurden ursprünglich 25-polige Sub-D-Stecker verwendet. Da hierbei die meisten Leitungen reine Drucker- bzw. Steuerleitungen sind, die für die meisten Peripheriegeräte nicht benötigt werden, wurde der neunpolige Sub-D-Stecker eingeführt. Die RS232-Schnittstelle arbeitet seriell mit je einer Datenleitung für beide Übertragungsrichtungen, d.h. die Bits werden, im Gegensatz zur parallelen Datenübertragung, nacheinander auf einer Leitung übertragen. Die hierfür benötigte Seriell-Parallel-Wandlung geschieht in UARTs (Universal Asynchronous Receiver Transmitter), entweder als integriertes Modul in einem Mikrocontroller oder als Stand-Alone-Baustein, im Folgenden SIO genannt.

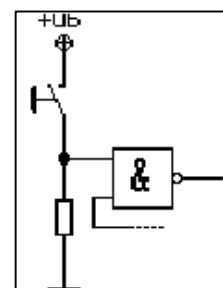
3.1.4 Pull-Up-Widerstände

Ein Pull-Up-Widerstand zieht die elektrische Spannung bis zum Betriebsspannungswert $+U_b$. Will man dafür sorgen, dass der Eingangspin logisch LOW erhält wenn die Taste gedrückt wird, liegt der Taster bzw. Schalter zwischen GND und dem Logik-Baustein. Der Pull-Up-Widerstand liegt in diesem Fall zwischen dem Eingang und $+U_b$. Beim Öffnen des Tasters zieht der Pull-Up-Widerstand die Spannung am Anschlusspin hoch bis zum Betriebsspannungswert $+U_b$, was logisch HIGH entspricht.



3.1.5 Pull-Down-Widerstände

Ein Pull-Down-Widerstand zieht die elektrische Spannung auf GND, oder im Falle einer symmetrischen Betriebsspannung von $\pm U_b$, auf $-U_b$ hinunter. Will man dafür sorgen, dass der Eingangspin logisch HIGH erhält wenn die Taste gedrückt wird, liegt der Taster bzw. Schalter zwischen dem Eingang des Gatters und $+U_b$. Der Pull-Down-Widerstand liegt zwischen dem Eingang und GND. Beim Öffnen des Kontaktes zieht der Pull-Down-Widerstand die



Spannung am Eingang hinunter auf GND, was logisch LOW entspricht. Bei der Anwendung von Pull-Up- und Pull-Down-Widerständen geht es immer darum, dass bei einem offenen Kontakt der korrekte logische Eingangspegel garantiert ist.

3.1.6 RAM (Random Access Memory)

Ein RAM ist ein Schreib- / Lesespeicher. In Speicherschaltungen mit größerer Kapazität verwendet man heute fast ausschließlich dynamische RAM. Sie sind meistens bitorganisiert und müssen zum Erreichen der geforderten Wortbreite in entsprechender Anzahl zusammengeschaltet sein. Durch den großen zusammenhängenden Adressbereich kann die Kaskadierung mehrerer Bausteine über Moduladressen in vielen Fällen entfallen. Für kleinere Systeme bevorzugt man statische RAM, da diese kein „Refresh“ benötigen und bei einem Ausfall der Versorgungsspannung mit Akkus kleiner Kapazität gepuffert werden können. Bei gemischtem Betrieb mit Festwertspeichern muss der Zentralspeicher funktionsbedingt aus Modulen bestehen. Der große, zusammenhängende Adressbereich von dynamischen RAM`s kann nicht unterteilt werden und erfordert dann eine Decodierlogik zum Ein- / Ausblenden von RAM und Festwertspeicher.

3.1.7 ROM (Read Only Memory)

Ein ROM ist ein Speicher, bei dem die Daten durch eine Maske fest „verdrahtet“ werden. ROM-Speicher verwendet man, wenn keine nachträglichen Änderungen mehr zu erwarten sind. Sie werden vorwiegend als Programmspeicher eingesetzt. Speicheraufbau und Organisation sind vergleichbar mit statischen RAM`s.

3.2 Software

3.2.1 Assemblersprache

Eine Assemblersprache ist eine spezielle Programmiersprache, welche die Maschinensprache einer spezifischen Prozessorarchitektur in einer für den Menschen lesbaren Form repräsentiert. Jede Computerarchitektur hat folglich ihre eigene Assemblersprache. Ein Programm in Assemblersprache wird auch als Assemblercode bezeichnet. Es wird durch einen speziellen Compiler in direkt ausführbare Maschinensprache (Maschinencode) umgewandelt. Die umgekehrte Umsetzung von Maschinencode in Assemblercode wird Disassemblierung genannt. Allerdings lassen sich dabei wesentliche Informationen wie Bezeichner und Kommentare nicht wiederherstellen, da diese bereits bei der Kompilierung unwiederbringlich verloren gingen. Programme in Assemblersprache zeichnen sich dadurch aus, dass man die komplette Bandbreite des Computers ausnutzen und Hardwarechips direkt programmieren kann. Weil Assemblerprogramme faktisch auf Maschinencode-Ebene arbeiten, sind sie immer ganz erheblich kleiner und deshalb schneller als Programme, die einen ähnlichen Grad an Komplexität aufweisen, aber Hochsprachen als Basis haben.

Die Nachteile von Assemblerprogrammen sind u.a. höhere Fehleranfälligkeit (durch erhöhte Komplexität), damit verbunden ein sehr großer Programmieraufwand bei umfangreichen Projekten.

3.2.2 Programmiersprache ‚C‘

C ist eine prozedurale Programmiersprache, die auf fast allen Computersystemen zur Verfügung steht. Im Gegensatz zu z.B. BASIC gibt es seit der Definition von ANSI C relativ einheitliche Implementierungen auf den verschiedenen Plattformen. In jedem C-System mit Laufzeitumgebung steht auch die genormte Standard C Library zur Verfügung. Konzeptionell ist C auf einfache Kompilierbarkeit angelegt, die Sprache wird daher manchmal auch als „high level assembler language“ bezeichnet. In C geschriebene Programme zeichnen sich im Vergleich zu manchen anderen Hochsprachen durch eine sehr gute Performance aus. Obwohl C nicht objektorientiert ist und mit C++, Java oder C# auf C basierende, objektorientierte Weiterentwicklungen mit ungleich umfangreicheren Bibliotheken zur Verfügung stehen, ist C nach wie vor weit verbreitet. Die Kerne fast aller heute verbreiteten Betriebssysteme wurden in C implementiert.

3.2.3 Compiler

Ein Compiler (auch Kompilierer oder Übersetzer) ist ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm in ein semantisch äquivalentes Programm einer Zielsprache umwandelt. Üblicherweise handelt es sich dabei um die Übersetzung eines von einem Programmierer in einer Programmiersprache geschriebenen Quelltextes in Assemblersprache, Bytecode oder Maschinensprache. Die Anwendung eines Compilers wird als Kompilierung bezeichnet.

3.2.4 Linker

Unter einem Linker oder Binder versteht man ein Programm, das einzelne Programmmodule zu einem ausführbaren Programm zusammenstellt. Es wird zwischen statischem und dynamischem Linken unterschieden.

3.2.4.1 Statisches Linken

Das statische Linken ist ein Vorgang, der typischerweise während der Entwicklung des Programms erfolgt, so dass der Benutzer ein fertig zusammengesetztes Programm erhält. Dies besteht dann bei vollständig statisch gelinkten Programmen aus einer einzelnen Datei.

3.2.4.2 Dynamisches Linken

Es ist möglich, das Auflösen der Funktions- und Variablennamen zu verschieben, bis das Programm tatsächlich ausgeführt wird. In diesem Falle spricht man von "dynamically linked library" (DLL) oder "shared library". Dies hat den Vorteil, dass man die Bibliothek nachträglich leicht austauschen kann, die Programme kleiner werden, und dass der Speicher nur einmal benötigt wird, wenn mehrere Programme dieselbe Bibliothek verwenden. Der Nachteil besteht darin, dass man irgendwie sicherstellen muss, dass auch die richtige DLL in der richtigen Version installiert ist.

3.2.5 Interrupt

Unter Interrupt versteht man die kurzfristige Unterbrechung eines Programms durch eine von der CPU abzuarbeitende Befehlssequenz, die Interrupt Service Routine (ISR). Anschließend wird das Programm an der unterbrochenen Stelle fortgesetzt. Sinn eines Interrupts ist es, schnell auf Ein- / Ausgabe-Signale (z.B. Tastatur etc.) oder Zeitgeber (Timer) zu reagieren. Interrupts sind notwendig, um auf zeitkritische Ereignisse reagieren zu können. Ausgelöst werden Interrupts durch so genannte Interrupt Controller mittels eines Interruptrequests (IRQ), einer Signalisierung an die CPU. Wird ein Interrupt von der CPU angenommen, wird in einer festen Sequenz, dem Interruptzyklus, die auslösende Quelle des Interrupts über den Datenbus identifiziert, dadurch der zugehörige Interruptvektor gefunden und der Sprung in die Interrupt-Service-Routine ausgelöst. Stehen zu diesem Zeitpunkt mehrere IRQs von unterschiedlichen Quellen an, so wird durch den Interrupt-Controller der Vektor des wichtigsten Interrupts durch ein Prioritätsverfahren bestimmt und angenommen. Hardware-Interrupts sind normalerweise maskierbare Interrupts, deren Auslösung durch spezielle Instruktionen unterbunden (maskiert) werden kann. Für Sonderfälle, wie z.B. Stromausfall, werden zusätzlich auch nicht maskierbare Interrupts (NMI) implementiert, die immer einen Sprung des Prozessors in die Interrupt-Routine auslösen, wobei ohne besondere Maßnahmen diese NMI's auch durch sich selbst unterbrochen werden können. Das ist auch der Grund, warum normale Hardware-Interrupts beim Sprung in die Interrupt-Service-Routine die Interrupts automatisch maskieren. Durch Rückspeicherung des Statusregisters wird beim Rücksprung immer der alte Zustand wiederhergestellt.

Hardware-Interrupts sind gegenüber dem unterbrochenen Programm immer asynchron, d.h. sie treten zu keinem bestimmten Zeitpunkt auf. Daher dürfen Interrupts ohne besondere synchronisierende Maßnahmen keinen direkten Einfluss auf Programme ausüben.

Ablauf eines Interrupts:

- Hardware bestimmt den Interruptvektor des IRQs mit der höchsten Priorität
- Hardware sichert Befehlszähler und Statusregister auf dem Stack
- Hardware holt neuen Befehlszähler vom Interruptvektor
- Software der Interrupt-Service-Routine sichert auch alle selbst benutzten Register auf dem Stack, da ansonsten die Daten des unterbrochenen Tasks überschrieben würden.
- Interrupt-Service-Routine läuft

- Bei Bedarf kann nach Aufruf einer speziellen Betriebssystemfunktion durch die Interrupt-Service-Routine einen entsprechenden Task durch den Scheduler des Betriebssystems gestartet werden. Da das einige Zeit dauert, kann evtl. zwischenzeitlich der gleiche Interrupt erneut auftreten, was im Algorithmus der Interrupt-Service-Routine zu berücksichtigen ist.
- Software der Interrupt-Service-Routine rücksichert alle selbst gesicherten Register.
- Interrupt-Service-Routine sichert sich durch Rücksprung (Rückspeichern von Befehlszähler und Statusregister vom Stack, der dadurch wieder seinen alten Stand wie vor der Unterbrechung hat, so als wäre nichts gewesen).
- Der aufgerufene Task kann die weitere Bearbeitung der gepufferten Daten übernehmen.

3.2.6 Begrifflichkeiten der Informatik

In den späteren Beschreibungen der einzelnen Module werden u.a. gängige Begriffe aus der Informatik verwendet, die hier zuvor noch einmal kurz erläutert werden.

Werden die Begriffe „reset“ oder „zurückgesetzt“ verwendet, so meint dies lediglich, dass der Inhalt einer Variable 0 ist. Treten die Ausdrücke „set“ oder „gesetzt“ auf, wird darunter der Inhalt 1 verstanden.

Des Weiteren gibt es die Möglichkeit, Werte miteinander zu verknüpfen. Das Zeichen „|“ steht für eine ODER-Verknüpfung, während das „&“ für eine UND-Verknüpfung steht.

Sowohl hinter den Begrifflichkeiten „Steuerwort“, als auch hinter „Datenwort“ sowie „Daten“ verbirgt sich in der Regel eine Information im Hexadezimalcode.

4 Hardware

4.1 Z80 – EMUF – Platine

4.1.1 Konstruktionsüberlegungen

Aufgrund der Entscheidung eine neue EMUF-Platine mit serieller Schnittstelle der Firma „Kanis“ zu bestellen, mussten wir uns überlegen, wie wir möglichst zeit- und kostengünstig diese Platine für unsere zusätzlich benötigte Peripherie modifizieren, ohne die Platine oder Bausteine zu ändern. Das Hauptproblem war die Adressierung unserer Zusatzperipherie.

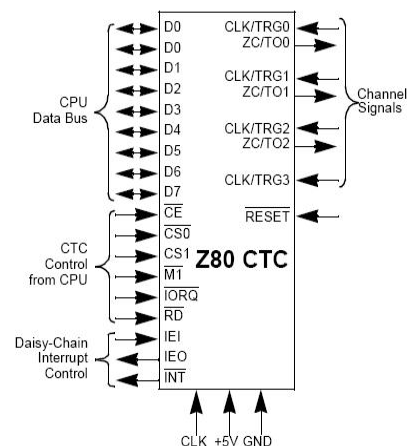
Da unser System mit über elf Peripheriebausteinen relativ komplex ist, bräuchte man einen Adressdecoder, der von vier auf sechzehn kodieren kann. Glücklicherweise war aber noch eine Ausgangsleitung des vorhandenen Adressdecoders frei, die wir zum Kaskadieren eines zweiten Adressdecoders nur für unsere Zusatzperipherie nutzen konnten. Somit haben wir wiederum den Originalzustand der EMUF-Platine erhalten und nur ein einzelnes Signal umgeleitet.

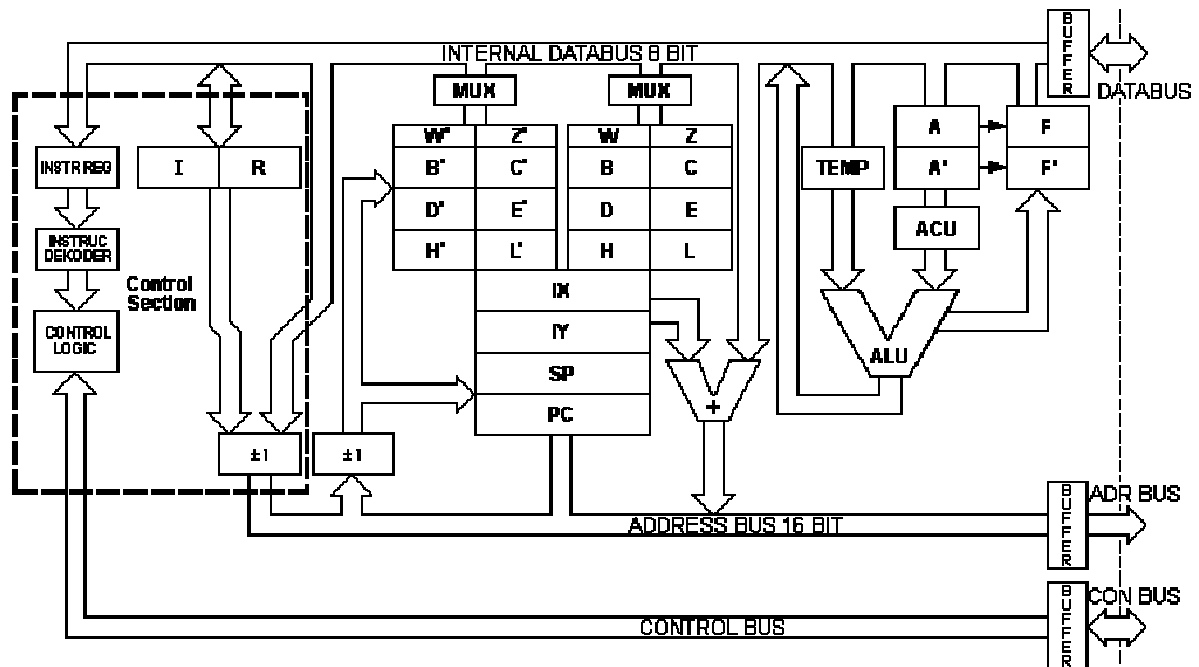
(siehe Kapitel Adressdecoder)

4.1.2 Aufbaubeschreibung

Ein Mikroprozessorsystem besteht immer aus der Prozessoreinheit, hier der Zilog Z80, dem ROM als Festwertspeicher für das Programm, dem RAM als dynamischer Zwischenspeicher, dem Adressdecoder zum Ansteuern der Peripheriebausteine und meistens zusätzlich mindestens einem Ein- / Ausgabebaustein, der PIO, sowie einer SIO als serielle Schnittstelle. Alle Komponenten sind mit dem Systembus verbunden, worüber die Komponenten mit der Prozessoreinheit kommunizieren können.

In unserem Fall verfügt die Kanis-EMUF-Platine über zwei PIO-Bausteine und eine universelle SIO.





Aufbau der Z80- CPU

4.1.2.1 CPU (Central-Prozessing-Unit)

Die Central-Prozessing-Unit, kurz CPU, besteht grob gesehen aus dem Rechenwerk, dem Befehlswerk und dem internen Speicher. Über das Bus-Interface greift die CPU auf den Systembus zu.

4.1.2.2 ALU (Arithmetic Logic Unit)

Die ALU ist der eigentliche Rechner. In ihr werden alle arithmetischen und logischen Funktionen und Berechnungen ausgeführt. Zur ALU gehören auch der Akku (Speicher) und die Flags (Ereignisspeicher).

4.1.2.3 CU (Control Unit)

In der Control Unit befindet sich das Befehlsregister, das alle Befehle enthält, die der Mikroprozessor ausführen kann. Hier werden auch die Befehle dekodiert. Dann gibt

es noch eine zeitliche und logische Steuerung, die auf die ALU bei Rechenoperationen zugreift. Von hier werden auch der Steuerbus, die Interrupts und die Serielle Ein- und Ausgabe gesteuert.

4.1.2.4 Interner Speicher (Akku)

Zum internen Speicher gehören zum einen wichtige Register, die als Zwischenspeicher dienen, als auch der Befehlszähler, in welchem die Speicherzelle steht, aus welcher der nächste Befehl geladen wird.

4.1.2.5 Ablauf einer Befehlsausführung

- Der Befehlszählerinhalt wird auf den Adressbus gelegt
- Über den Datenbus wird der Operations-Code des Befehls ins Befehlsregister geladen
- Der Befehlszähler wird erhöht
- Der Befehl wird decodiert
- Wenn nötig (bei Mehrbytebefehlen) werden weitere Bytes des Befehls in das Befehlsregister geholt
- Der Befehl wird ausgeführt

Quellcode		Assembler		Programmspeicher
IN 01 INC A OUT 02 ... Quellcode in mnemonischer Schreibweise			1000 1110 0110 1100 Programm in Maschinensprache (einzelne Bits)	

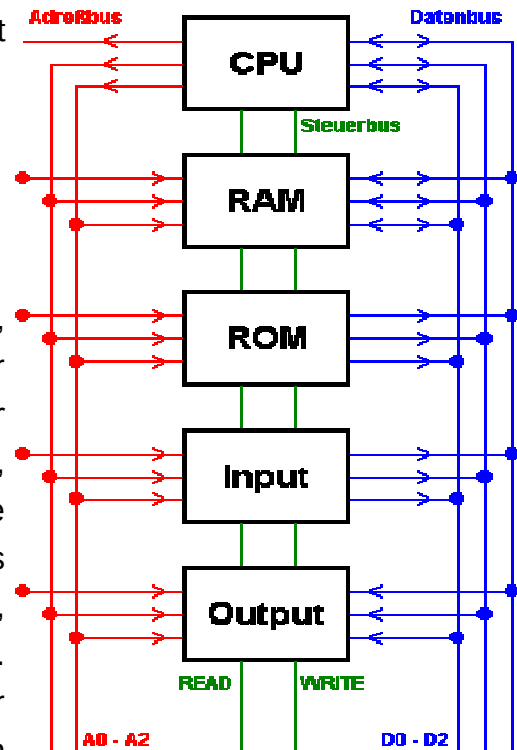
Programmierung eines Mikroprozessors

4.1.2.6 Systembus

Der Systembus eines Mikroprozessorsystems teilt sich in Datenbus, Adressbus und Steuerbus auf.

4.1.2.6.1 Datenbus

Zur Übertragung von Daten zwischen Prozessor, Arbeitsspeicher und Peripherie steht der Datenbus zu Verfügung. Die Anzahl der Datenbusleitungen bestimmt die Anzahl der Bits, die pro Takt übertragen werden können. Alle Baugruppen, die Daten an den Datenbus übernehmen oder übergeben können (Prozessor, Speicher, I/O) sind mit dem Datenbus verbunden. Die Datenfreigabe erfolgt durch den Prozessor jeweils nur für eine Baugruppe über einen Adressdecoder.



4.1.2.6.2 Adressbus

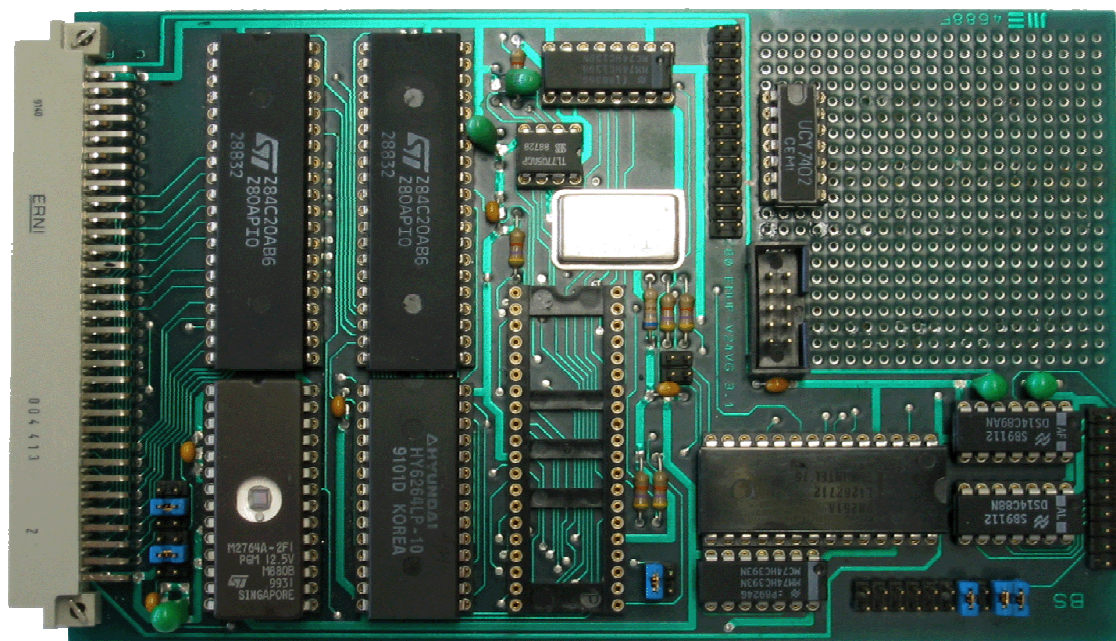
Der Adressbus ist für die Übertragung von Speicheradressen zuständig. Über den Adressbus wird jedoch nicht nur der Speicher, sondern auch die Peripherie-Geräte angesprochen. Die Anzahl der Adressleitungen ist ein entscheidender Faktor für die Anzahl der maximal zu adressierenden Speicherplätze, bzw. Peripheriebausteine.

$$\text{maximal adressierbare Speicherplätze} = 2^{\text{Anzahl der Adressleitungen}}$$

Diese Formel dient zur Ermittlung der maximal nutzbaren Speichergröße, die ein Prozessor ansprechen kann (in Byte).

4.1.2.6.3 Steuerbus

Der Steuerbus dient der Übertragung bestimmter Steuersignale an die einzelnen Komponenten. Auf diese Weise wird der auf dem Adressbus angesprochenen Komponente mitgeteilt, was sie zu tun hat, so z.B. die Daten vom Datenbus aufzunehmen oder an den Datenbus zu übergeben. Der Steuerbus beinhaltet auch die Interrupt-Leitungen über die die Peripherie-Geräte dem Prozessor eine Unterbrechungsanforderung signalisieren können.



4.1.3 Zusätzliche Änderungen

Als eine der Voraussetzungen unseres Projektes müssen die im RAM gespeicherten Datensätze auch ohne Spannungsversorgung erhalten bleiben, damit sie auch nach einem Einsatz mit leerem Akku auslesbar sind. Dazu wurde der enthaltene Standard-RAM gegen einen gepufferten RAM ausgetauscht, der die Daten bis zu zehn Jahre statisch speichert. Um ein effektives Arbeiten zu ermöglichen, wurde kein Standard EPROM mit 8Kbyte verwendet, sondern ein EEPROM mit 32Kbyte. Somit ist eine schnelle Programmänderung problemlos möglich.

Der vorhandene SIO-Baustein benötigt in der Regel eine zusätzliche, separate Spannungsversorgung von $\pm 12V$, um das serielle Signal auf V.24-Standard zu bringen. Da unser komplettes System jedoch mit einer Spannungsversorgung über einen Akku und einen Festspannungsregler auf +5V läuft, haben wir keine $\pm 12V$ zur Verfügung. Deshalb müssen wir diese Spannung aus den vorhandenen +5V erzeugen. Am einfachsten lässt sich dies mit einem integrierten DC-DC-Wandler realisieren, der in dem MAX232 bereits enthalten ist. Dieser Baustein ist für die Signalerzeugung auf den V.24-Standard optimiert und liefert $\pm 10V$ an Pin 2 und Pin 6 zu Pin 15 als Masse. Um aber wiederum diverse Verdrahtungsänderungen auf der Kanis-EMUF-Platine zu verhindern, änderten wir nicht die Ausgänge der SIO, sondern benutzen den MAX232 als reine Spannungsquelle für die Versorgung der beiden der SIO nachgeschalteten IC's (IC11- IC12), die die V.24-Signale erzeugen.

Die Hauptspannungsversorgung des gesamten Mikroprozessorsystems wurde mit einem Festspannungsregler auf der PIO-Verteiler-Platine realisiert und über den Stecker ST04 auf der KANIS-EMUF-Platine „rückwärts“ eingespeist.

Als Hauptfunktion übernimmt die Z80-EMUF-Platine die kompletten Programm- und Steuerfunktionen für das ganze System inklusive der Peripheriebausteine.

Die beiden bereits vorhandenen PIO's sind hauptsächlich für die Tasten- und LED-Ansteuerung zuständig und sind wie nebenstehend belegt. Zusätzlich ist noch ein Ausgang für die LCD-Hintergrundbeleuchtung belegt. Alle Taster sind hardwareseitig Low-Aktiv definiert. Alle Anderen Eingänge oder Ausgänge sind hardwareseitig High-Aktiv definiert.

PIO 2

DisTas1:

Taster 1= A0	LED 1= B1
Taster 2= B0	LED 2= A2
Taster 3= A1	LED 3= B2

DisTas2:

Taster 1= A3	LED 1= B4
Taster 2= B3	LED 2= A5
Taster 3= A4	LED 3= B5

DisTas3:

Taster 1= A6	LED 1= B7
Taster 2= B6	
Taster 3= A7	

PIO 1

DisTas3:

LED 2= B0	LCD-Bel. = B1
LED 3= A0	

Taster	→ Low- Aktiv
LED	→ High- Aktiv
LCD- Bel.	→ High- Aktiv
Buzzer	→ High- Aktiv

4.1.3.1 Jumperbelegungen für ROM und RAM

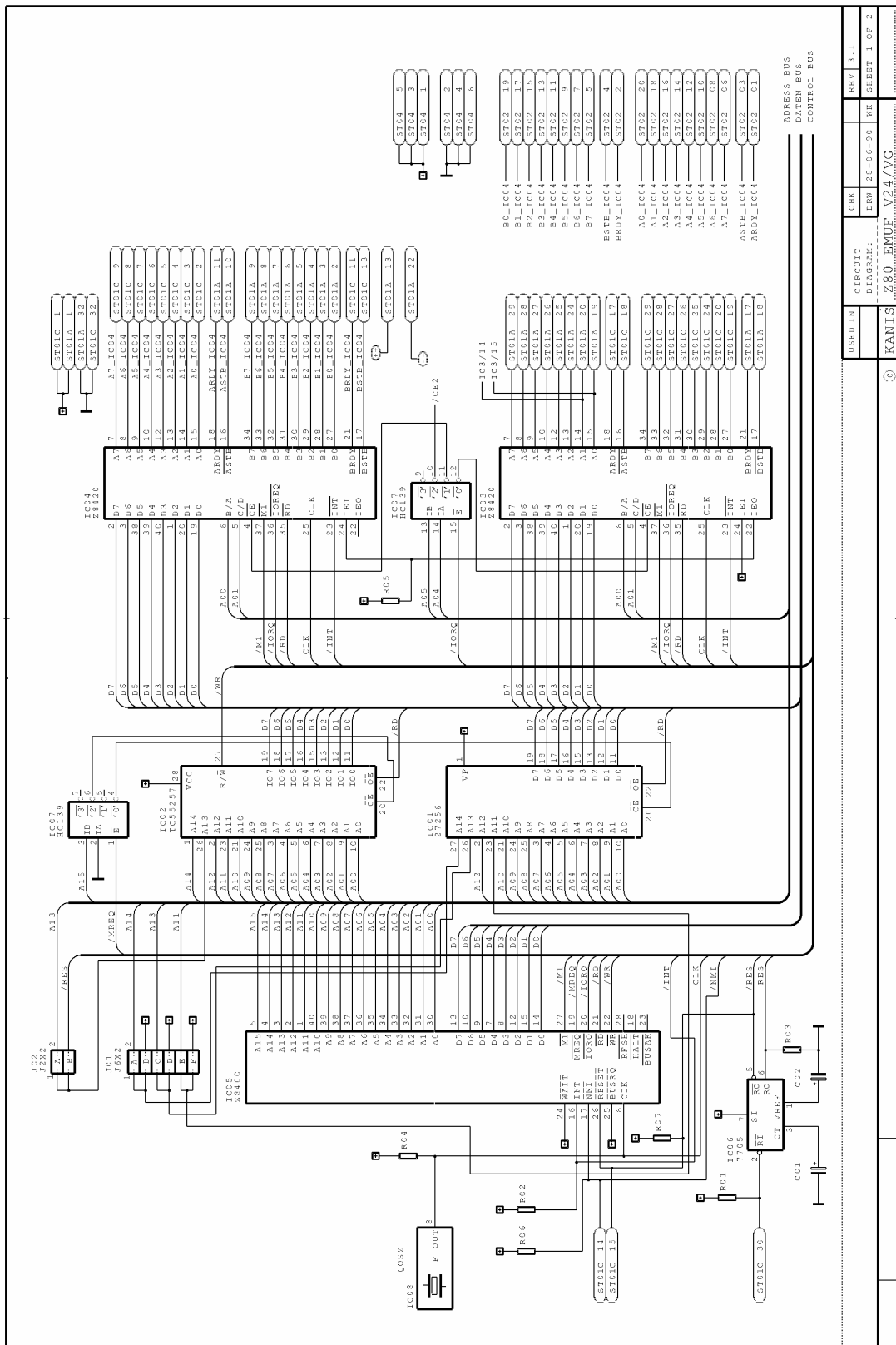
EPROM	J1A	J1B	J1C	J1D	J1E	J1F
2716	O	O	O	Z	O	Z
2732	O	O	O	Z	Z	O
2764	O	Z	O	O	Z	O
27128	O	Z	Z	O	Z	O
27256	Z	O	Z	O	Z	O

RAM	J2A	J2B
8K	O	Z
32K	Z	O

Z=zu

O=offen

4.1.4 Schaltplan Z80-EMUF-Platine



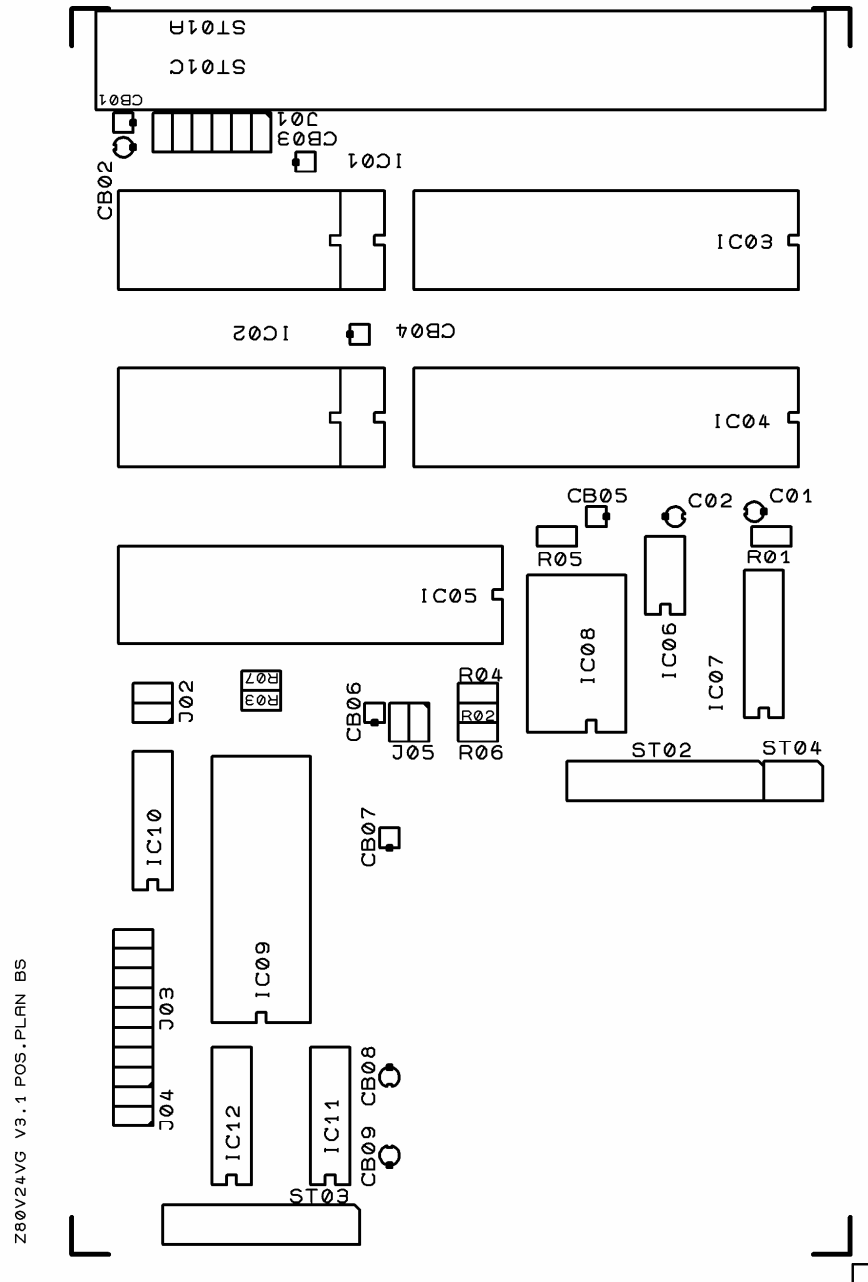


4.1.5 Bestückungsplan Z80-EMUF-Platine

Bestückungsplan Bauteilseite, V3_10, Z80V24VG.PLC, 11.11.96

I:\PJ\Z80V24VG\LAY\V3_10\PL0T_PS\

>> FG <<



4.1.6 Adressdecoder

Im Blockschaltbild kann man erkennen, dass einige Adressleitungen und die Datenleitungen D0-D7 zu allen Peripheriebausteinen führen. Wird nun z.B. von der CPU ein Datenwort auf den Datenbus und gleichzeitig die Adresse 8003 (A15, A0 und A1=H) auf den Adressbus gelegt, dann wären alle Peripheriebausteine adressiert und könnten die Daten übernehmen, bzw. würden Datenkollisionen verursachen.

Damit nur ein Baustein die Daten übernehmen kann, ist eine zusätzliche Selektierung notwendig. Diese Aufgabe übernimmt der Adressdecoder 74HC139. Die Decoderausgänge sind mit den entsprechenden Freigabe- bzw. Selektierungseingängen (CE / CS) der Peripheriebausteine verbunden.

Aufgrund der Ansteuerung von insgesamt elf unabhängigen Peripheriebausteinen benötigen wir einen Adressdecoder der 2^4 Bausteine adressieren kann. Um jedoch nicht die vorhandene Kanis-EMUF-Platine zu ändern, haben wir die Adressdecoder kaskadiert. Somit erhalten wir die Originalfunktionen auf der Platine und können einen zusätzlichen Adressdecoder separat auf unserer Zusatzhardware realisieren. Dafür wird nur der noch unbenutzte Decoderausgang vom vorhandenen Decoder (74HC139, Pin 9) auf der Platine umgeleitet und als Freigabesignal für den zweiten Adressdecoder auf dem PIO-Verteiler an CE angeschlossen.

Mit der Adressleitung A15 und dem Freigabesignal /MREQ am Adressdecoder 74HC139 / 1 wird entsprechend der zwei Zustände der Adressleitung (High / LOW) der RAM oder der ROM selektiert.

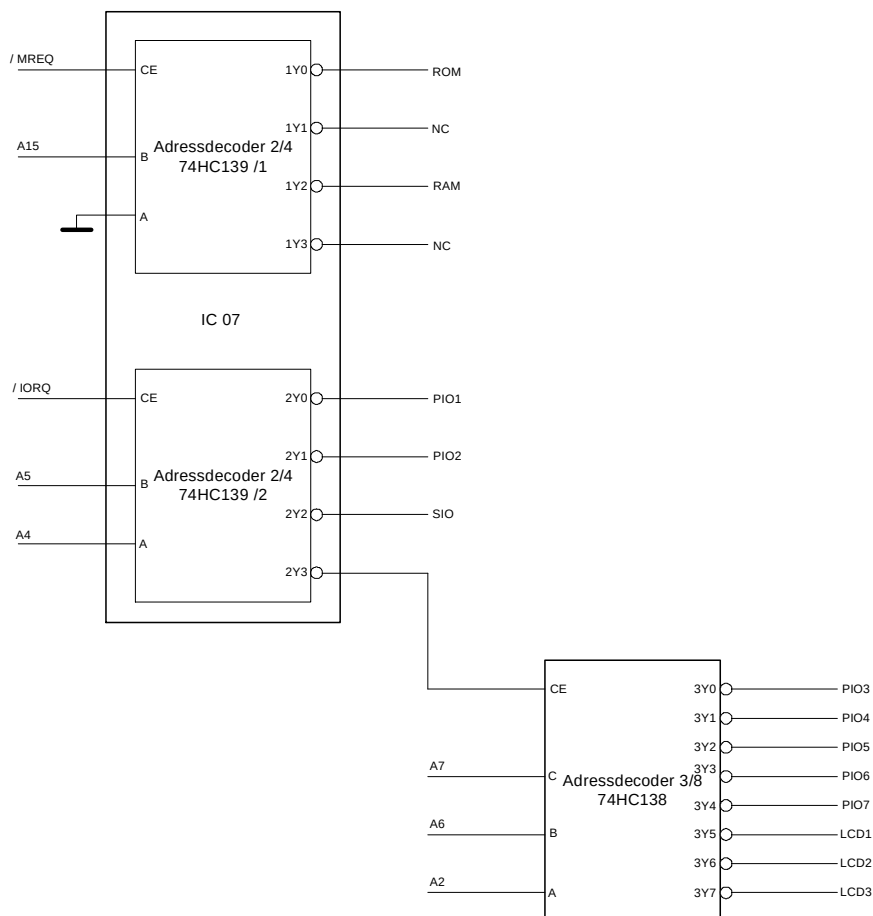
Mit den Adressleitungen A4 und A5, sowie dem Freigabesignal /IORQ am Adressdecoder 74HC139 / 2 werden entsprechend der anliegenden Adresse die Peripheriebausteine PIO1, PIO2, SIO, oder der zusätzliche Adressdecoder über den Decoderausgang 2Y3 angesteuert.

Die Adressleitungen A2, A6, A7 und die Freigabeleitung 2Y3 steuern den kaskadierten Adressdecoder 74HC138, dessen Ausgänge die PIO 3-7, sowie die LCD 1-3 ansteuern.

Im Folgenden ist die komplette Funktionstabelle der kaskadierten Decoder abgebildet.

INPUTS 74HC139 /2			OUTPUTS 74HC139 /2				INPUTS 74HC138				OUTPUTS 74HC138							
CE	A5	A4	2Y0	2Y1	2Y2	2Y3	CE	A7	A6	A2	3Y0	3Y1	3Y2	3Y3	3Y4	3Y5	3Y6	3Y7
H	X	X	H	H	H	H	H	X	X	X	H	H	H	H	H	H	H	H
L	L	L	L	H	H	H	H	X	X	X	H	H	H	H	H	H	H	H
L	L	H	H	L	H	H	H	X	X	X	H	H	H	H	H	H	H	H
L	H	L	H	H	L	H	H	X	X	X	H	H	H	H	H	H	H	H
L	H	H	H	H	H	L	L	L	L	L	L	H	H	H	H	H	H	H
L	H	H	H	H	H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	H	H	H	H	L	L	L	H	L	H	H	L	H	H	H	H	H
L	H	H	H	H	H	L	L	L	H	H	H	H	H	L	H	H	H	H
L	H	H	H	H	H	L	L	H	L	L	H	H	H	H	L	H	H	H
L	H	H	H	H	H	L	L	H	L	H	H	H	H	H	H	L	H	H
L	H	H	H	H	H	L	L	H	H	L	H	H	H	H	H	H	L	H
L	H	H	H	H	H	L	L	H	H	H	H	H	H	H	H	H	H	L

4.1.6.1 Blockschaltbild Adressdecoder



4.1.6.2 Steueradressen

Gerät		Steueradresse	
Baustein	Decoderausgang	Befehl	Hex
EEPROM	1Y2		ab 0000h
RAM	1Y0		ab 8000h

PIO 1	2Y0	Data Port A	00h
Taster/ LED		Data Port B	01h
		Control Port A	02h
		Control Port B	03h

PIO 2	2Y1	Data Port A	10h
Taster/ LED		Data Port B	11h
		Control Port A	12h
		Control Port B	13h

SIO	2Y2	Data	20h
		Control	22h

PIO 3	3Y0	Data Port A	30h
PID 1/2		Data Port B	31h
		Control Port A	32h
		Control Port B	33h

PIO 4	3Y1	Data Port A	34h
PID 3/4		Data Port B	35h
		Control Port A	36h
		Control Port B	37h

PIO 5	3Y2	Data Port A	70h
PID 5/6		Data Port B	71h
		Control Port A	72h
		Control Port B	73h

PIO 6	3Y3	Data Port A	74h
PID 7/8		Data Port B	75h
		Control Port A	76h
		Control Port B	77h

PIO 7	3Y4	Data Port A	B0h
PID 9		Data Port B	B1h
		Control Port A	B2h
		Control Port B	B3h

Display 1	3Y4	Control Write	B4h
Trupp 1		Control Read	B5h
		Data Write	B6h
		Data Read	B7h

Display 2	3Y5	Control Write	F0h
Trupp 2		Control Read	F1h
		Data Write	F2h
		Data Read	F3h

Display 3	3Y6	Control Write	F4h
Trupp 3		Control Read	F5h
		Data Write	F6h
		Data Read	F7h

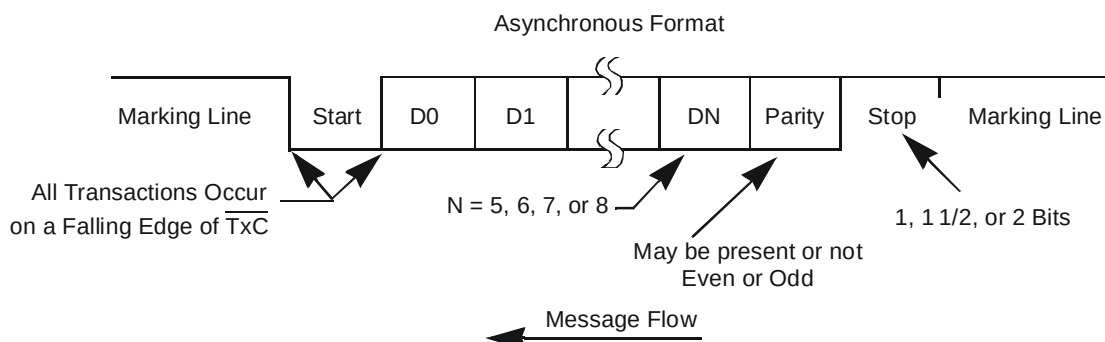
4.1.7 SIO

Die RS232-Schnittstelle ist eine Spannungsschnittstelle, d.h. die Informationen werden durch eine elektrische Spannung kodiert. Für die Datenleitungen (TxD und RxD) wird eine negative Logik verwendet, wobei eine Spannung zwischen -3V und -12V eine logische „1“ und eine Spannung zwischen +3V und +12V eine logische „0“ darstellt. Bei den Steuerleitungen (DCD, DTR, DSR, RTS, CRS, RI) wird der aktive Zustand durch eine Spannung zwischen +3V und +12V, der inaktive Zustand durch eine Spannung zwischen -3V und -12V dargestellt.

Es gibt zwei verschiedene Verfahren der Datenübertragung, die asynchrone und die synchrone Übertragung.

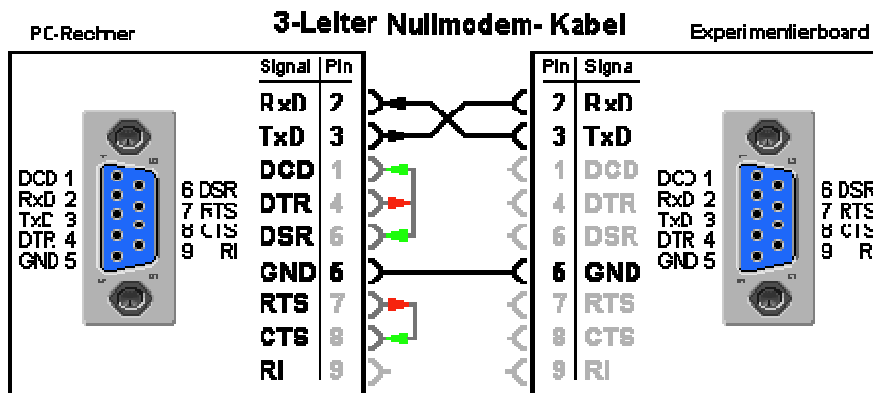
Bei der asynchronen Datenübertragung wird kein zusätzliches Synchronisations-Signal (Clock-Signal) übertragen. Die Synchronisation erfolgt ausschließlich über die Datenleitungen. Das geschieht, indem sich der Empfänger auf die erste negative Flanke nach einer Ruhepause synchronisiert und die folgenden Bits mit der Baudrate abtastet.

Damit das funktioniert, müssen die Baudraten von Sender und Empfänger übereinstimmen und jedes übertragene Byte muss von einer negativen Flanke eingeleitet werden. Letzteres wird durch Start- (0) und Stoppbits (1) sichergestellt. Die Ruhepause muss ausreichend lang sein, um zu gewährleisten, dass die erste negative Flanke von einem Startbit kommt. Zwischen den Start- und Stoppbits werden die Daten unverändert übertragen.

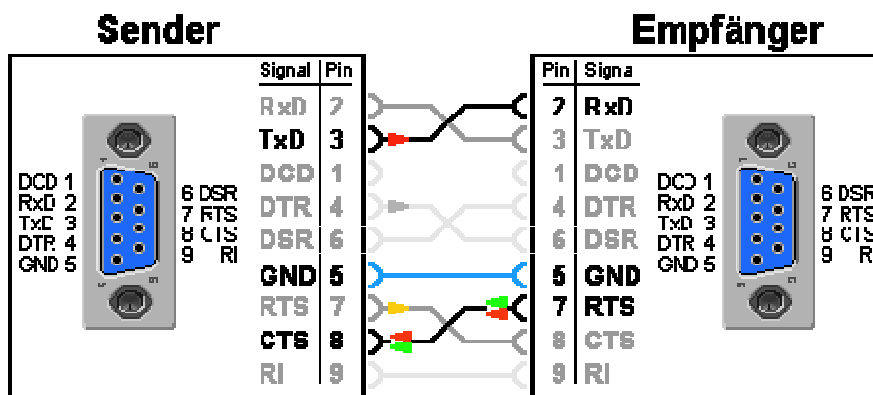


Um einen Datenverlust zu vermeiden, muss der Empfänger die Datenübertragung anhalten können, wenn eine Verarbeitung weiterer Daten nicht möglich ist. Das geschieht über so genannte Handshakes, von denen es zwei unterschiedliche Arten gibt, den Software- und den Hardwarehandshake.

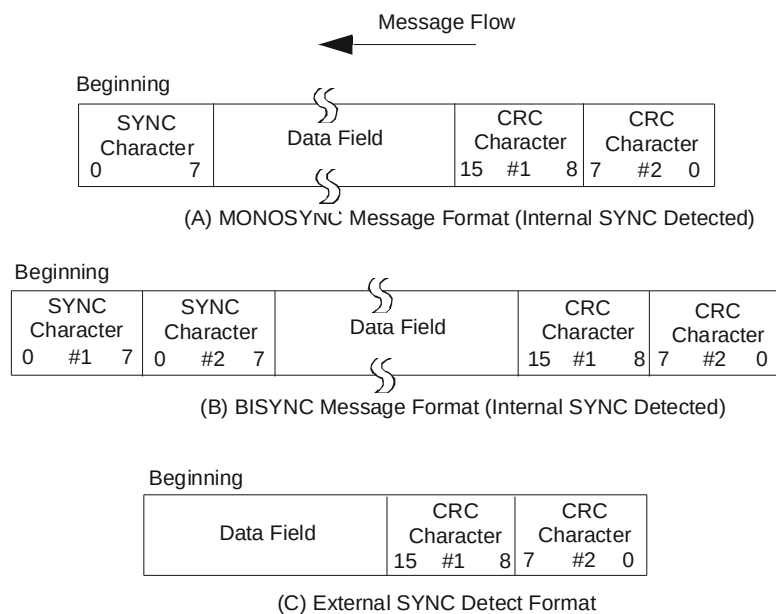
Beim Softwarehandshake sendet der Empfänger zur Steuerung des Datenflusses spezielle Zeichen an den Sender. Dementsprechend werden für die Datenübertragung lediglich drei Leitungen (RxD, TxD, Gnd) benötigt. Im Stecker müssen dafür RTS und CTS, DTR mit DSR und DCD gebrückt werden, weil sonst die Hardware wartet, da die Handshakeleitungen nicht richtig bedient werden.



Beim Hardwarehandshake steuert der RxD-Empfänger über Steuerleitungen die Handshake-Eingänge CTS, DSR und DCD des TxD-Senders mit seinen Handshake-Ausgängen RTS auf CTS und DTR auf DSR und DCD. Ein Hardwarehandshake besteht demzufolge aus 5 Leitungen (TxD, RxD, Gnd, RTS und CTS).



Das synchrone Verfahren ist im Prinzip eine Optimierung des asynchronen Verfahrens. Bei der synchronen Übertragung fügt man zur Synchronisation eine bestimmte Bitfolge (Synchronwort) ein, die sonst nicht auftreten kann. Auf diese Weise kann der Empfänger den Beginn eines Datenblocks erkennen. Wenn keine Daten vorliegen, werden nur Synchronworte gesendet, somit kann sich der Taktgenerator im Empfänger immer auf das ankommende Signal synchronisieren.



Eine Auflistung und Kurzbeschreibung der wichtigsten vorhandenen Signale einer RS232 Schnittstelle sind im Folgenden aufgelistet.

Abkürzung	Name	Beschreibung	Pin-Nr. 25-pol.	Pin-Nr. 9-pol.	Input/Output (vom PC)
	Common Ground	Gemeinsame Abschirmmasse (nicht Datenmasse)	Pin 1	n.c.	-
TxD	Transmitt Data	Leitung für gesendete Daten	Pin 2	Pin 3	Out
RxD	Receive Data	Leitung für Datenempfang	Pin 3	Pin 2	In
RTS	Request To Send	Eine logische 1 signalisiert der Gegenstelle, dass sie Daten senden kann	Pin 4	Pin 7	Out
CTS	Clear To Send	Eine logische 1 signalisiert der Gegenstelle, dass sie Daten annehmen kann	Pin 5	Pin 8	In
DSR	Dataset ready	Eine logische 1 signalisiert dem PC Einsatzbereitschaft	Pin 6	Pin 6	In
GND	Ground	Signalmasse	Pin 7	Pin 7	-
DCD	Data Carrier Detected	Gerät signalisiert dem PC, dass eingehende Daten erkannt werden	Pin 8	Pin 1	In
DTR	Data Terminal Ready	PC signalisiert Betriebsbereitschaft. Dadurch kann ein Gerät eingeschaltet oder zurückgesetzt werden.	Pin 20	Pin 4	Out
RI	Ring Indicator	Gerät zeigt dem PC, dass ein Anruf ankommt	Pin 22	Pin 9	In

Namen und Bedeutungen der wichtigsten Signalleitungen. Leitungsbezeichnungen immer aus Sicht des DTEs (PC).

4.2 PIO-Verteiler-Platine

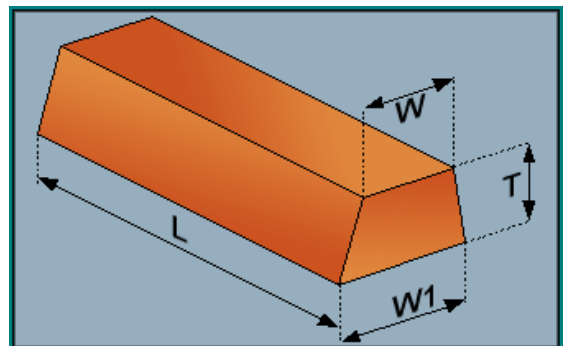
4.2.1 Konstruktionsüberlegungen

Als eine Herausforderung galt es eine Platine zu entwickeln, die fünf zusätzliche PIO-Bausteine aufnehmen kann, die Hauptspannungsversorgung übernimmt, den kaskadierten Adressdecoder enthält und ferner alle Verteilerstecker von den Display-Platinen, den PID-IN-Platinen, dem Systembus sowie die Taster- / LED-Signale durchleitet. Dies sollte möglichst kompakt auf einer doppelseitigen Euroformat-Platine realisiert werden. Da diese Platine von vornherein sehr komplex werden würde, stand die Entscheidung fest diese Platine nicht selbst zu ätzen und zu bestücken. Wir haben zwar eine doppelseitige Musterplatine erfolgreich geätzt, jedoch wäre das Fehlerrisiko sowie der Zeitfaktor beim Erstellen einer doppelseitigen Platine mit über 350 Durchkontaktierungen zu groß, so dass wir die Platine in einem professionellen PCB-Produktionslabor erstellen ließen. Somit konnten wir wiederum Fertigungsfehler und Probleme bei der späteren Inbetriebnahme minimieren, vorausgesetzt, dass das Layout korrekt war. Denn aufgrund dieser Entscheidung hatten wir keine Möglichkeit eventuelle Layoutfehler anhand einer Musterplatine zu beheben.

Bei der Platinenentwicklung mit dem Layouteditor EAGLE mussten wir besonders darauf achten, dass die Leiterbahnen für die Spannungsversorgung zu den einzelnen Komponenten vom Querschnitt groß genug ausgelegt werden, da wir insgesamt über elf Hauptkomponenten verwenden. Um eine aussagefähige Querschnittsberechnung durchführen zu können, mussten wir zunächst den Gesamtstromverbrauch des Systems ermitteln. Dafür entnahmen wir den Datenblättern jeweils die max. Stromaufnahme der Hauptkomponenten und erlangten somit folgendes Ergebnis:

Je Display Platine ca. 120 mA, für die gesamten fünf PIO's auf der Verteiler-Platine insgesamt ca. 300 mA, und für die Z80-EMUF-Platine ca. 350 mA (laut Hersteller).

Somit errechneten wir einen maximalen Gesamtstromverbrauch des Systems von ca. 1000mA. Um nun den genauen Querschnitt zu errechnen nahmen wir uns das Programm AMPTRACK sowie das Programm POLAR TRC-100 zur Hilfe (siehe Anhang), womit man den Leiterbahnquerschnitt in Abhängigkeit von der max. Strombelastbarkeit und der Temperaturerhöhung, sowie den Leiterbahnwiderstand berechnen kann.



Auf Basis dieser Berechnungen und einer Kupferbeschichtung von 35 µm entschieden wir uns bezüglich der Spannungsversorgung für eine Leiterbahnbreite von 0,4014 mm, die bei einer Temperaturänderung ΔT 20°C bis zu 1,55 A durchleiten kann. Dies ist bei einer theoretisch max. Stromaufnahme von 1 A eine ausreichende Dimensionierung. Leider konnten wir keine doppelte Sicherheit im Sinne einer breiteren Leiterbahn bis zu 2 A realisieren, da die Platine dafür zu komplex ist und die Leiterbahnen sehr kompakt geroutet wurden.

Bei unserem Praxistest haben wir eine maximale Stromaufnahme inklusive der Displayhintergrundbeleuchtung, die etwa die Hälfte des Gesamtstromverbrauches ausmacht, von ca. 650 mA gemessen. Somit erreichen wir im durchschnittlichen Normalbetrieb gerade die Hälfte des maximal möglichen Stromes der berechneten Leiterbahn. Dies kommt daher, dass niemals alle vorhandenen Komponenten ihre maximal mögliche Stromaufnahme zur gleichen Zeit anfordern.

In der folgenden Tabelle ist eine grobe Übersicht von Leiterbahnstärken in Abhängigkeit von der Umgebungstemperatur zu sehen (Achtung: nur Richtwerte!).

Schicht- stärke <i>Thickness copper layer</i>	Leiterbahn- breite <i>Track- width</i>	Max. Strom in Abhängigkeit zur Temperaturerhöhung. <i>Max. current depending on temperature increase.</i>				
		10 °C	20 °C	30 °C	45 °C	60 °C
35 µm	0,25 mm	0,5 A	0,8 A	1,0 A	1,3 A	1,6 A
	0,50 mm	1,0 A	1,6 A	2,0 A	2,5 A	3,0 A
	1,00 mm	2,2 A	3,0 A	3,6 A	4,2 A	4,8 A
	1,50 mm	3,0 A	3,8 A	4,6 A	5,3 A	6,5 A
	2,00 mm	3,8 A	5,0 A	6,5 A	7,5 A	8,5 A
	3,00 mm	4,5 A	6,5 A	8,0 A	9,5 A	11,0 A
	4,00 mm	6,0 A	8,5 A	10,0 A	12,0 A	13,5 A
	5,00 mm	7,0 A	10,0 A	12,0 A	14,5 A	16,0 A
	6,00 mm	7,5 A	11,0 A	14,0 A	16,0 A	18,0 A
	8,00 mm	9,0 A	14,0 A	17,0 A	20,0 A	22,5 A
	10,00 mm	10,0 A	16,0 A	20,0 A	23,0 A	26,0 A
70 µm	0,25 mm	1,0 A	1,6 A	2,0 A	2,5 A	3,0 A
	0,50 mm	2,0 A	2,8 A	3,5 A	4,0 A	4,5 A
	1,00 mm	3,5 A	4,7 A	5,8 A	6,8 A	8,0 A
	1,50 mm	4,5 A	6,2 A	7,5 A	9,0 A	10,5 A
	2,00 mm	6,0 A	8,5 A	10,0 A	12,0 A	13,5 A
	3,00 mm	7,5 A	11,0 A	14,0 A	16,0 A	18,0 A
	4,00 mm	9,0 A	13,5 A	17,0 A	19,0 A	22,0 A
	5,00 mm	10,0 A	15,0 A	19,0 A	23,0 A	25,0 A
	6,00 mm	11,0 A	18,0 A	22,0 A	26,0 A	28,0 A

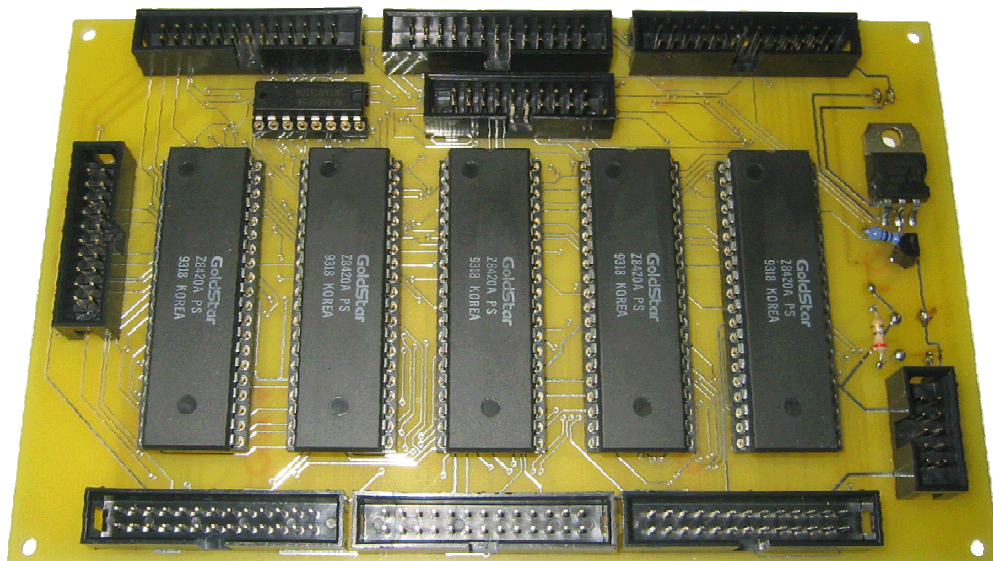
4.2.2 Aufbaubeschreibung

Wie bereits erwähnt, enthält die PIO-Verteiler-Platine fünf zusätzliche PIO-Bausteine zur Erweiterung des Mikroprozessorsystems für weitere IO-Peripherie. In diesem Fall besteht die IO-Peripherie hauptsächlich aus den PID-Sticks. Die Hauptfunktion dieser Platine besteht also darin, als Benutzereingabeschnittstelle zu agieren. Zusätzlich werden alle Signalverbindungen zwischen der CPU und sämtlicher Zusatzhardware über Flachbandkabel und Steckverbinder über die PIO-Verteiler-Platine geführt, die somit als zentraler Hauptverteiler im System dient.

Je 7 parallele Datenleitungen von den neun PID-Sticks (siehe Kapitel PID-Sticks) werden über die PID-IN-Platine direkt an Port A und B der PIO's durchgeleitet. Somit wird also jeweils ein PID-Stick einem Port einer PIO zugeordnet.

ID-Stick:	PID 1	PID 2	PID 3	PID 4	PID 5	PID 6	PID 7	PID 8	PID9
PORT:	PIO 3A	PIO 3B	PIO 4A	PIO 4B	PIO 5A	PIO 5B	PIO 6A	PIO 6B	PIO 7A

Die letzte PIO im System (PIO7) ist nur auf Port A von einem PID-Stick belegt. Port B ist also noch für zusätzliche Signale frei, wie z.B. dem des Alarmsignalgebers (Buzzer) und des Tasters für die Displayhintergrundbeleuchtung. Da die maximale Strombelastung der IO-Ports auf ca. 4 mA begrenzt ist, mussten wir bei allen Ausgangskomponenten die mehr als 4 mA benötigen, einen Treiberbaustein zwischenschalten. Der Buzzer hat eine Stromaufnahme von ca. 10 mA, folglich haben wir einen Feldeffekt-Transistor BS170 als Treiber vorgeschaltet, der als Schalter arbeitet. Der Taster ist wie alle Anderen Low-Aktiv und benötigt somit einen Pull-Up-Widerstand am Eingang der PIO, um in Ruheposition einen definierten Zustand zu liefern.

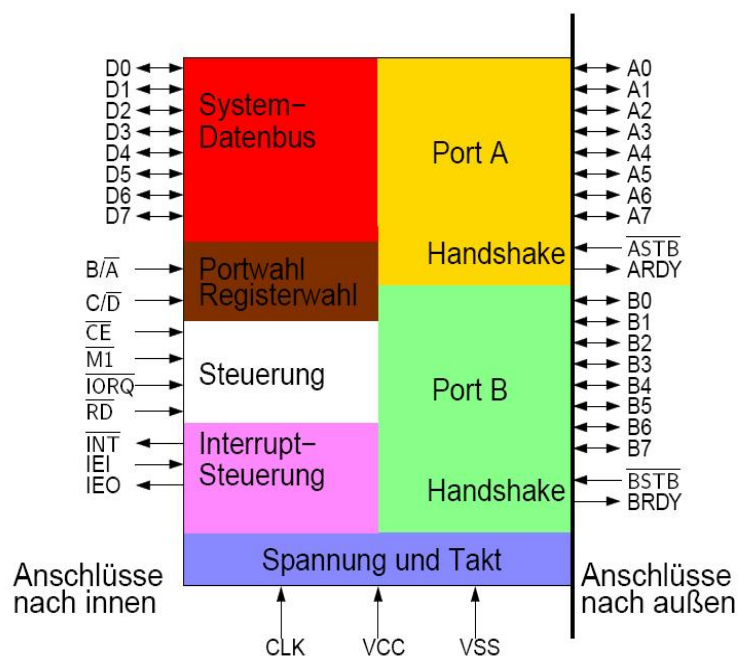


Die Spannungsversorgung für das komplette System von 5V wurde mit einem Very-Low-Drop Festspannungsregler verwirklicht, der mit einer Dropout-Spannung (Differenz zwischen Eingangs- und Ausgangsspannung) von max. 500mV minimale Verbrauchswerte aufweist. Somit können wir das System mit einem Akku von 6V betreiben. Der bereits erwähnte, kaskadierte Adressdecoder liefert von der PIO-Verteiler-Platine das Aktivierungssignal Chip-Enable (CE -> Low-Aktiv) für alle zusätzlichen PIO-Controller sowie allen Displays.

4.2.2.1 PIO-Controller

Die PIO-Controller selbst organisieren den parallelen Datentransfer zwischen Mikrocontroller und Peripherie. Die Aktivitäten des PIO-Controllers werden ausschließlich durch die CPU gesteuert. Somit ist es notwendig, dass beide Bausteine miteinander Signale austauschen können.

- D0 bis D7 werden mit dem System-Datenbus verbunden
- B/A und C/D werden mit den Leitungen A1 bzw. A0 des System-Adressbusses verbunden, um nebeneinander liegende I/O-Adressen für den PIO-Controller zu schaffen
- Port A und B dienen der Kommunikation mit der Peripherie. Verbleibende Anschlüsse werden mit den entsprechenden Leitungen des Systems verbunden.



Die PIO-Betriebsarten werden je Port festgelegt:

- Mode 0: Byteweise-Ausgabe im Handshake-Betrieb
- Mode 1: Byteweise-Eingabe im Handshake-Betrieb
- Mode 2: Byteweise-Ein-/Ausgabe im Handshake-Betrieb (bidirektional)
- Mode 3: Bitweise-Ein-/Ausgabe mit Datenrichtungssteuerung (bidirektional)

In unserem System wird mit Mode 3 gearbeitet, da wir die meisten Ports bitweise als Eingabe- und gleichzeitig als Ausgabe-Interface benutzen, wie z.B. PIO1 und PIO2, die mit den Tastern als Eingabe und den LED`s als Ausgabe belegt sind. Die beiden Handshakeleitungen ASTB/ARDY bzw. BSTB/BRDY sind im Betriebsmodus 3 nicht relevant und somit in unserem System nicht belegt. Da alle Anschlüsse der der Z80-Familie zugehörigen Bausteine tristatefähig sind, können unbenutzte Leitungen einfach „in der Luft hängen“, es sei denn sie sind als Eingänge deklariert und brauchen somit einen definierten Zustand, also einen Pull-Up- oder Pull-Down-Widerstand. In unserem Fall sind die Digitalcodes der PID-Sticks als Low-Aktiv definiert, so dass wir je einen Pull-Up-Widerstand pro Eingang benötigen. Diese wurden durch Widerstandsdekaden direkt an den jeweiligen Ports der PIO`s realisiert und nicht sichtbar unter den PIO-Bausteinen eingebaut.

Da der Prozessor für mehrere periphere Geräte zuständig ist, empfängt er auch mehrere Interrupt-Signale, kann allerdings immer nur eines pro Zeiteinheit abarbeiten. Somit muss eine Priorisierung gewährleistet sein. Der Interrupt-Requestor (= das den Interrupt anfordernde Gerät) muss dem System sowohl Beginn, als auch Ende seiner CPU-Inanspruchnahme signalisieren.

IE1 (= Interrupt-Enable High): PIO wird berechtigt, Interruptvektor /INT an die CPU zu senden.

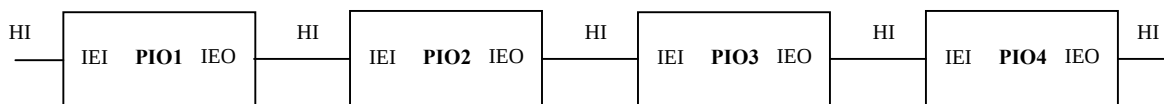
IE0 (= Interrupt-Enable Low): Aussendung des Interruptvektors wird in Anspruch genommen.

Wie bereits zuvor erwähnt, wird die Priorisierung der Interrupt-Requestor durch die so genannte Daisy-Chain realisiert.

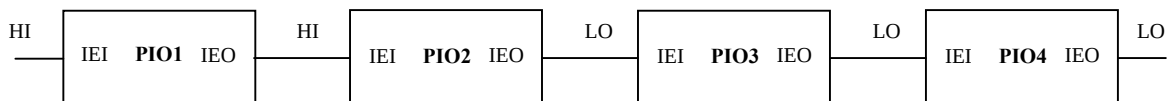
4.2.2.2 Daisy-Chain

Als „Daisy Chain“ wird eine Anzahl an Hardware-Komponenten bezeichnet, welche durch eine Datenleitung in Form einer Kette miteinander verbunden sind, sodass der Ausgang einer Hardwarekomponente mit dem Eingang der Nächsten verbunden ist. Die erste Komponente ist direkt mit der CPU verbunden. Wird nun ein Interrupt von einer beliebigen Komponente angefordert, wird dieser Interrupt erst dann von der CPU bearbeitet, wenn ein evtl. angeforderter Interrupt eines vorigen Interrupt-Requestors beendet wurde. Denn während der Bearbeitung eines Interrupts werden alle nachfolgenden Interruptanforderungen gesperrt, also von allen Komponenten die in der Daisy-Chain hardwaremäßig nachgeschaltet sind. Wichtig ist auch hier, dass die Daisy-Chain zwischen jeder Komponente auch mit einem Pull-Up-Widerstand versehen wird, da die IEI-Anschlüsse als Eingänge definiert sind.

1. Daisy Chain vor Interrupt-Auslösung:



2. Daisy Chain nach Interrupt-Auslösung:



4.2.2.3 Adressierung der PIO-Register

Der PIO-Controller ist in den I/O-Adressraum des Mikroprozessorsystems eingebunden und wird mit den Adressleitungen A0 an B/A zur Auswahl von Port A oder B, und A1 an C/D zur Auswahl des Steuer- oder Datenregisters angesteuert.

B/A	C/D	RD	adressiertes PIO-Register
0	0	0	Eingaberegister Port A
0	0	1	Ausgaberegister Port A
0	1	X	Steuerregister Port A
1	0	0	Eingaberegister Port B
1	0	1	Ausgaberegister Port B
1	1	X	Steuerregister Port B

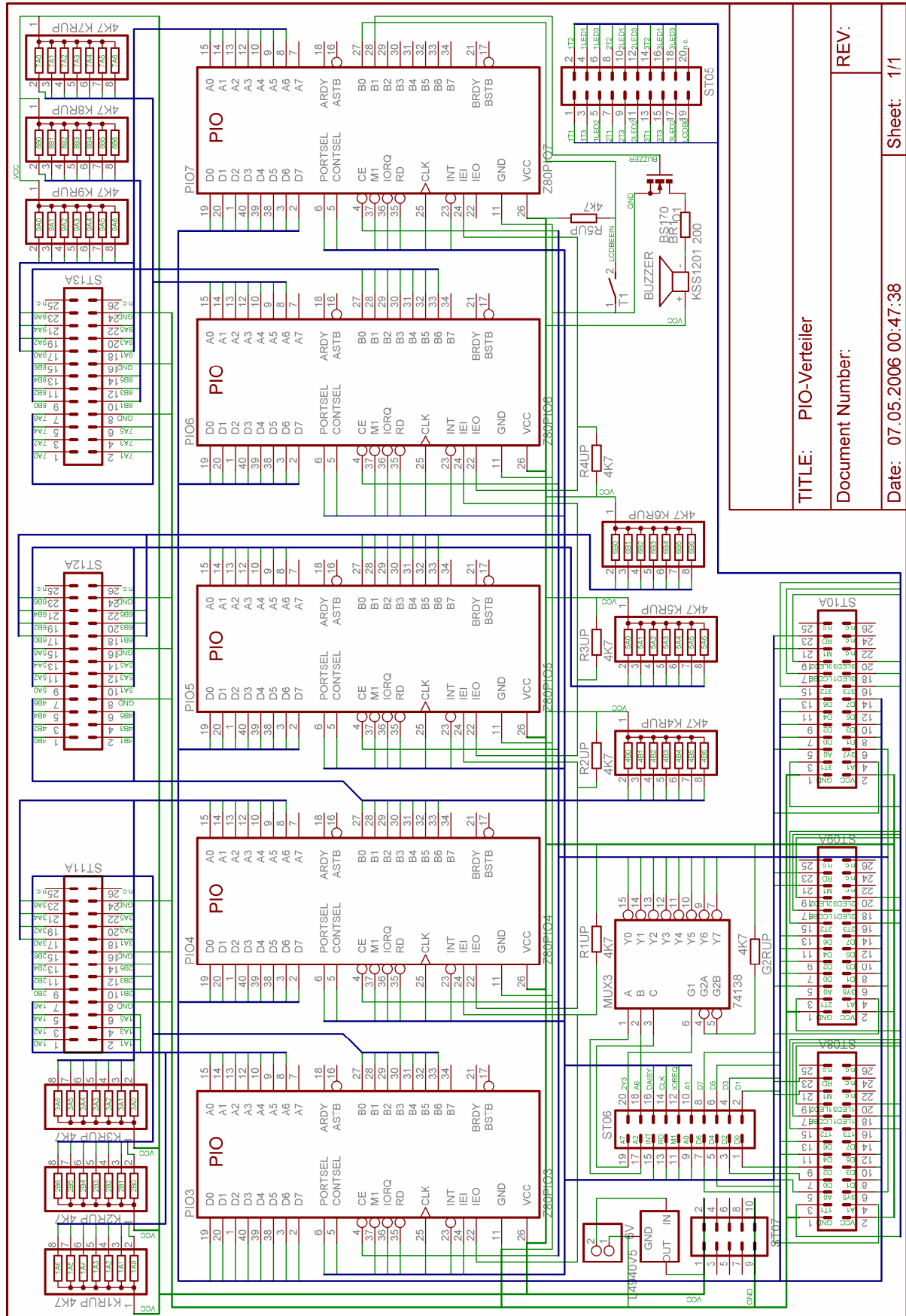
X = don't care

4.2.2.4 Initialisierung des PIO-Controllers

In der Initialisierungsphase wird der PIO-Controller auf die zu übernehmenden Aufgaben vorbereitet. Zur Initialisierung des PIO-Controllers muss die Steuerleitung C/D HIGH-Pegel führen.

- Es müssen immer 3 (PIO-Modi 0 bis 2), 4 oder 5 (PIO-Mode 3) Steuerwörter übergeben werden.
- Die Steuerwortfolge ist festgelegt:
 1. Betriebsarten-Steuerwort (B)
Im PIO-Mode 3 Datenrichtungs-Steuerwort (D) (obligatorisch)
 2. Interrupt-Vektor (V) für IM2 der CPU
 3. Interrupt-Steuerwort 8 (S)
Im PIO-Mode 3 Masken-Steuerwort (M) (optional)
- Die Steuerwörter (B), (V) und (S) besitzen Kennungsbits, um Verwechslungen auszuschließen.

4.2.3 Schaltplan PIO-Verteiler-Platine



TITLE: PIO-Verteiler

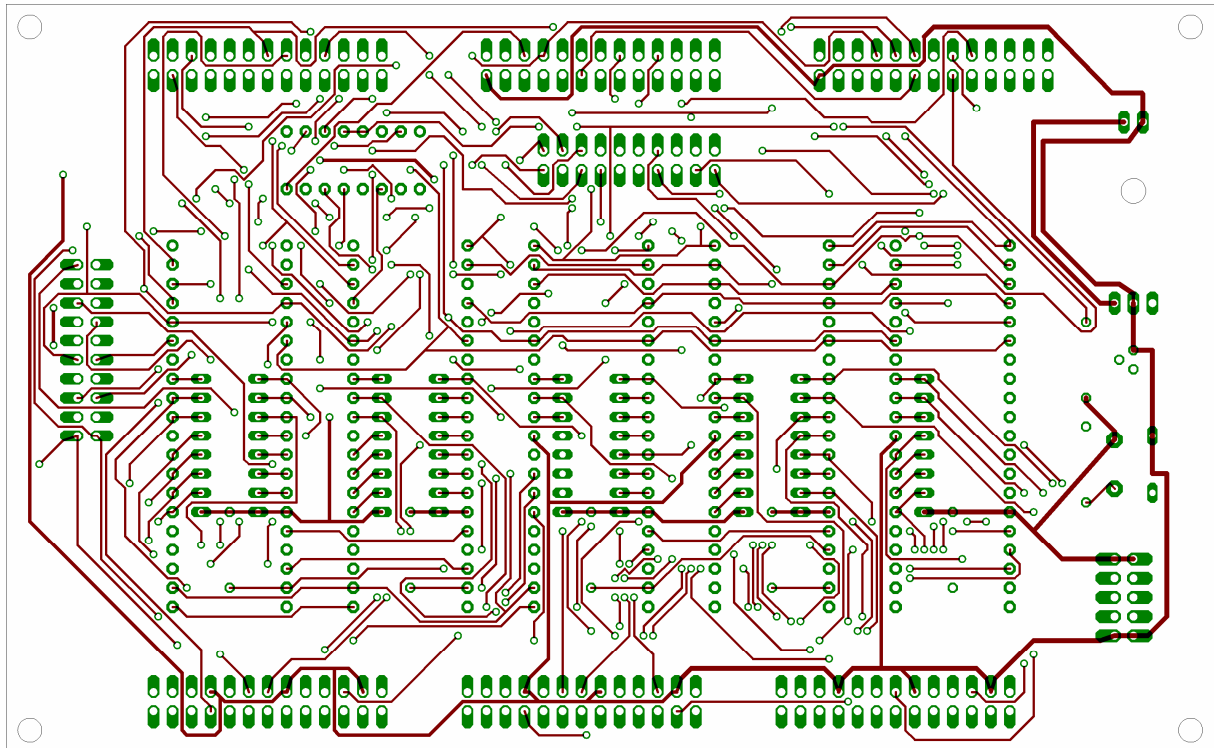
Document Number:

REV:

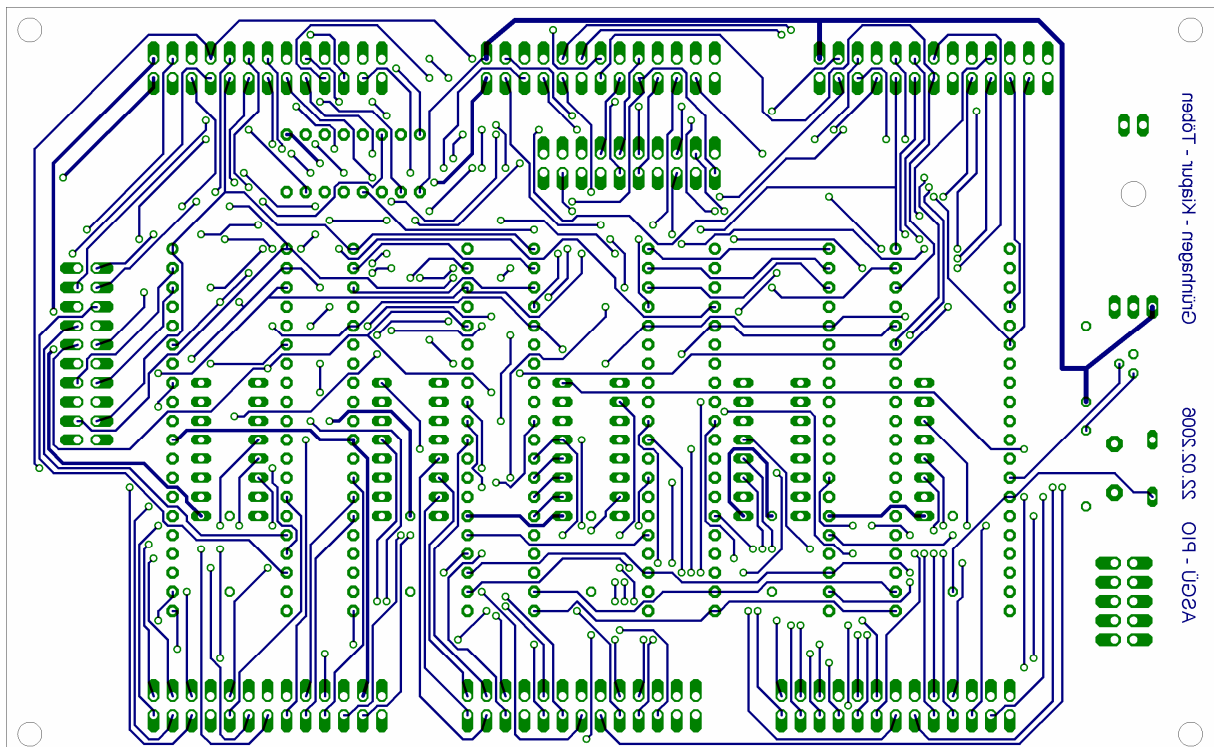
Date: 07.05.2006 00:47:38

Sheet: 1/1

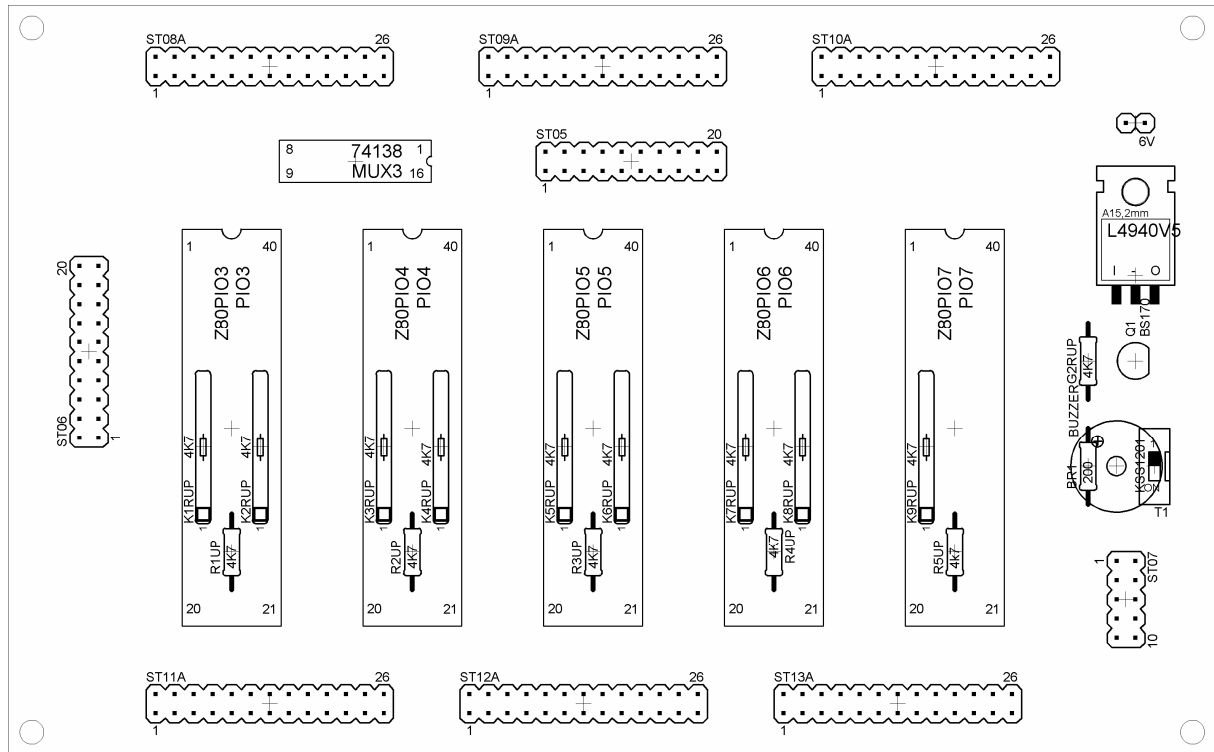
4.2.4 Layout PIO-Verteiler-Platine (Oberseite)



4.2.5 Layout PIO-Verteiler-Platine (Unterseite)



4.2.6 Bestückungsplan PIO-Verteiler-Platine



4.3 CPU-Extension-Platine

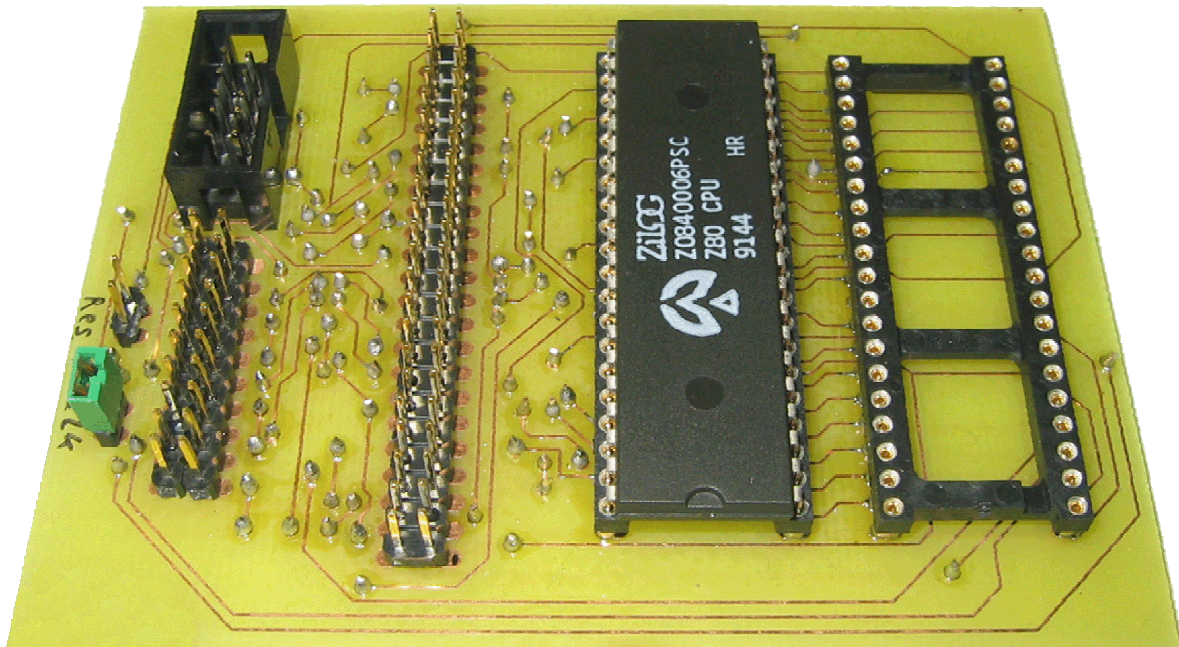
4.3.1 Konstruktionsüberlegungen

Normalerweise befindet sich die CPU auf der Z80-EMUF-Platine. Da wir jedoch das System um einige Elemente erweitert haben, mussten wir ebenso den Systembus an die erweiterten Elemente durchleiten, nur leider wurde der Systembus nicht auf einen der vielen Steckverbinder der EMUF-Platine hinausgeführt. Also mussten wir den Systembus auf der Platine irgendwie abgreifen. Dies konnte am einfachsten am Sockel der CPU durchgeführt werden, da hier sämtliche Signale zusammenführen. Ursprünglich hatten wir die CPU mit einem Sockel auf das noch freie Lochrasterfeld der Platine gelötet und über ein 40-poliges Flachbandkabel mit dem Originalsockel verbunden, sowie einen weiteren Stecker daneben gelötet, der mit den entsprechenden Signalleitungen des Systembusses der CPU verlötet war. Insgesamt wurde diese „Verdrahtung“ auf der Unterseite der Platine sehr unübersichtlich und erwies sich als sehr fehlerträchtig. Aus diesem Grund entschieden wir uns, eine zusätzliche Erweiterungsplatine zu entwickeln.

4.3.2 Aufbaubeschreibung

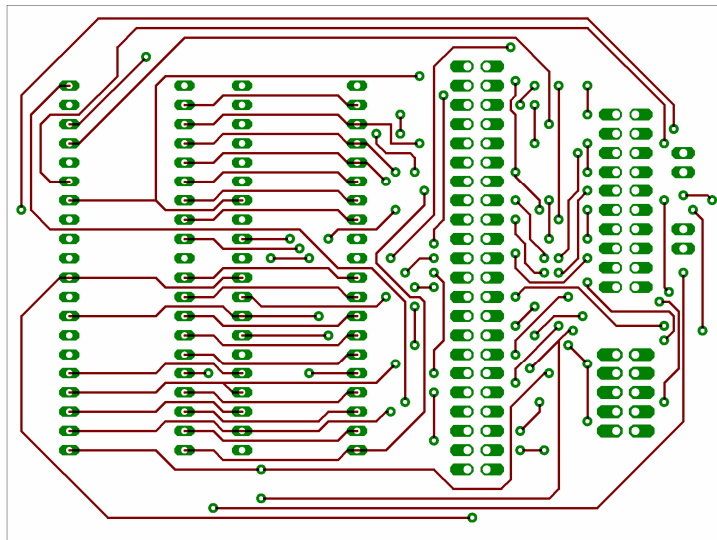
Die CPU-Erweiterungsplatine beinhaltet lediglich die CPU selbst, einen Sockel für das 40-polige Flachbandkabel zur Verbindung zur EMUF-Platine, einen Stecker für die Weiterleitung der benötigten Signale vom Systembus zu den PIO's und Displays, sowie einen 44-poligen Diagnosestecker mit allen Signalen des Systembusses. An dem Diagnosestecker kann man eine Bus-Test-Platine anschließen, um den kompletten Systembus und somit das Programm im Single-Step-Betrieb (ohne angeschlossenem Systemtakt des Quarzes) zu testen. Der Systemtakt des Quarzes und das Reset-Signal wurden beide jeweils über einen externen Jumper auf der Erweiterungsplatine hinausgeführt. Der Jumper CLK ist nur für die Inbetriebnahme und Testphase in Zusammenhang mit der Bus-Test-Platine im Single-Step Betrieb zu öffnen, im Normalbetrieb immer geschlossen zu halten. Mit dem Reset-Jumper löst man bei Kurzschluss einen Systemreset aus. Die CPU ist 1:1 mit dem Originalsockel verbunden. Zusätzlich wurden noch zwei wichtige Signale von der EMUF-Platine für die erweiterte Hardware „angezapft“ und über einen extra Stecker weitergeleitet. Das Ausgangssignal 2Y3 des Adressdecoders zum kaskadieren des zweiten Adressdecoders auf der PIO-Verteiler-Platine (siehe Kapitel Adressdecoder) und die Daisy-Chain von PIO2 zu PIO3 durchzuleiten

(siehe Kapitel Daisy-Chain).

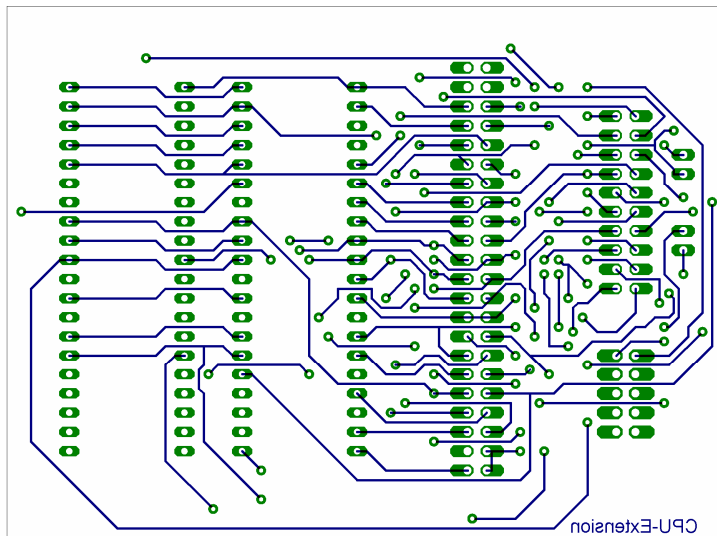


Der eigentliche Grund dieser Platine besteht also nur darin, den Systembus für die erweiterte Hardware durchzuleiten, da auf dem Lochrasterfeld der EMUF-Platine nicht genügend Platz vorhanden war, die Leitungen auf der Rückseite gut und fehlerfrei zu verlegen. Somit haben wir wiederum eine große Fehlerquelle minimiert und zusätzlich eine sehr hilfreiche Diagnoseschnittstelle hinzugefügt. Leider konnten wir uns aber auch auf der Erweiterungsplatine bei kompakter doppelseitiger Bauweise keine Durchkontaktierungen ersparen und mussten diese mit Hilfe eines Lötdrahtes verlöten, was bei hoher Anzahl von Durchkontaktierungen prinzipiell eine Fehlerquelle darstellt.

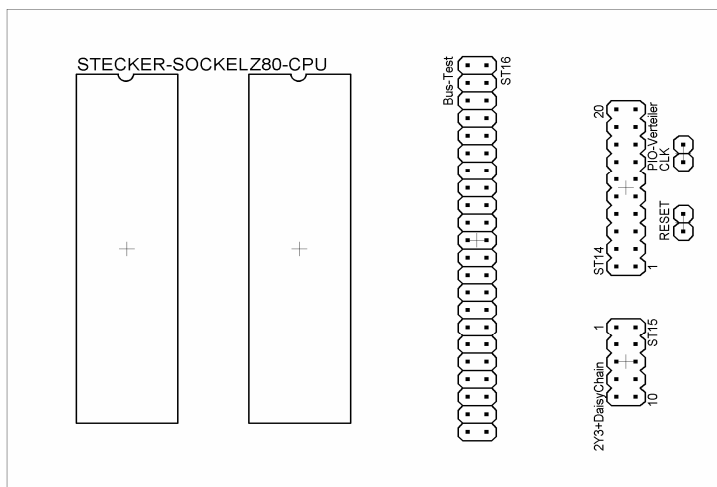
4.3.4 Layout CPU-Extension (Oberseite)



4.3.5 Layout CPU-Extension (Unterseite)



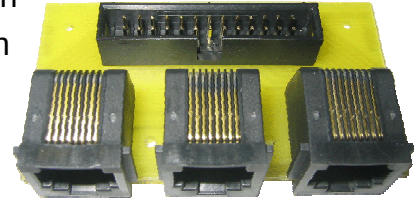
4.3.6 Bestückungsplan CPU-Extension



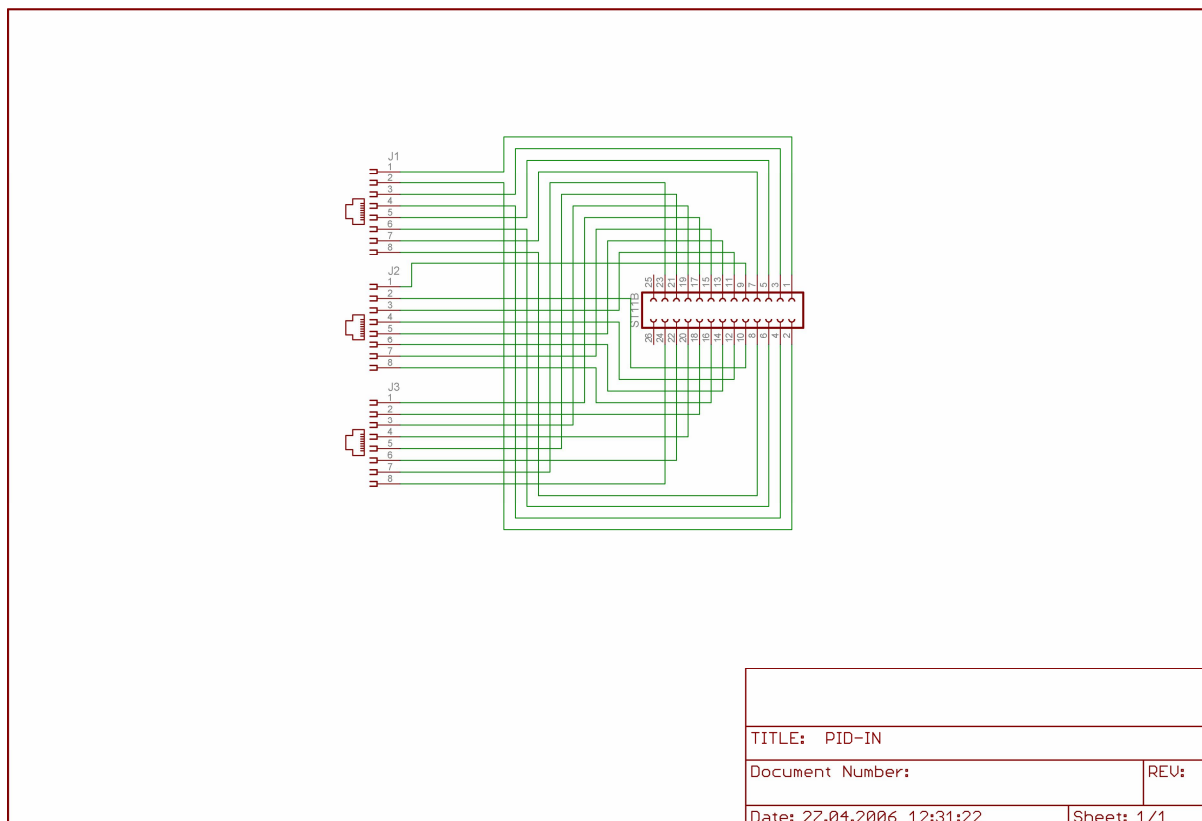
4.4 PID-IN-Platine

4.4.1 Aufbaubeschreibung

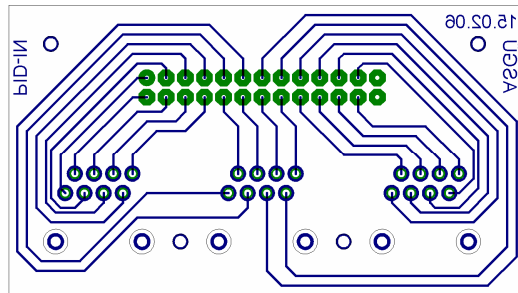
Die PID-IN-Platinen (Personen-ID-Input-Platinen) dienen lediglich der Aufnahme der Personen-ID-Sticks. In unserem System befinden sich drei dieser Platinen, welche mit jeweils drei Modular-Einbaubuchsen versehen sind. Daraus ergibt sich eine rein optische Trennung der einzelnen Trupps (jeweils drei Steckplätze pro Trupp). Des Weiteren befinden sich auf den Platinen 26-polige Pfostenstecker, welche mit den Einbaubuchsen verbunden sind. Über diese Pfostenbuchsen wird mittels Flachbandkabel die Verbindung zum PIO-Platine hergestellt. Wir haben die Variante mit drei einzelnen Platinen gewählt, um ein evtl. Austauschen im Fehlerfall schnell und kostengünstig zu gestalten.



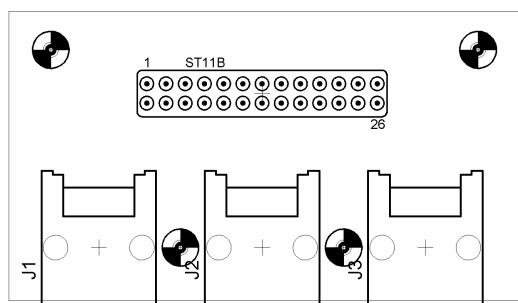
4.4.2 Schaltplan PID-IN-Platine



4.4.3 Layout PID-IN-Platine



4.4.4 Bestückungsplan PID-IN-Platine



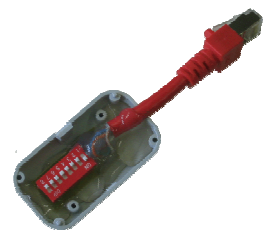
4.5 PID-Sticks (Personen-ID-Sticks)

4.5.1 Konstruktionsüberlegungen

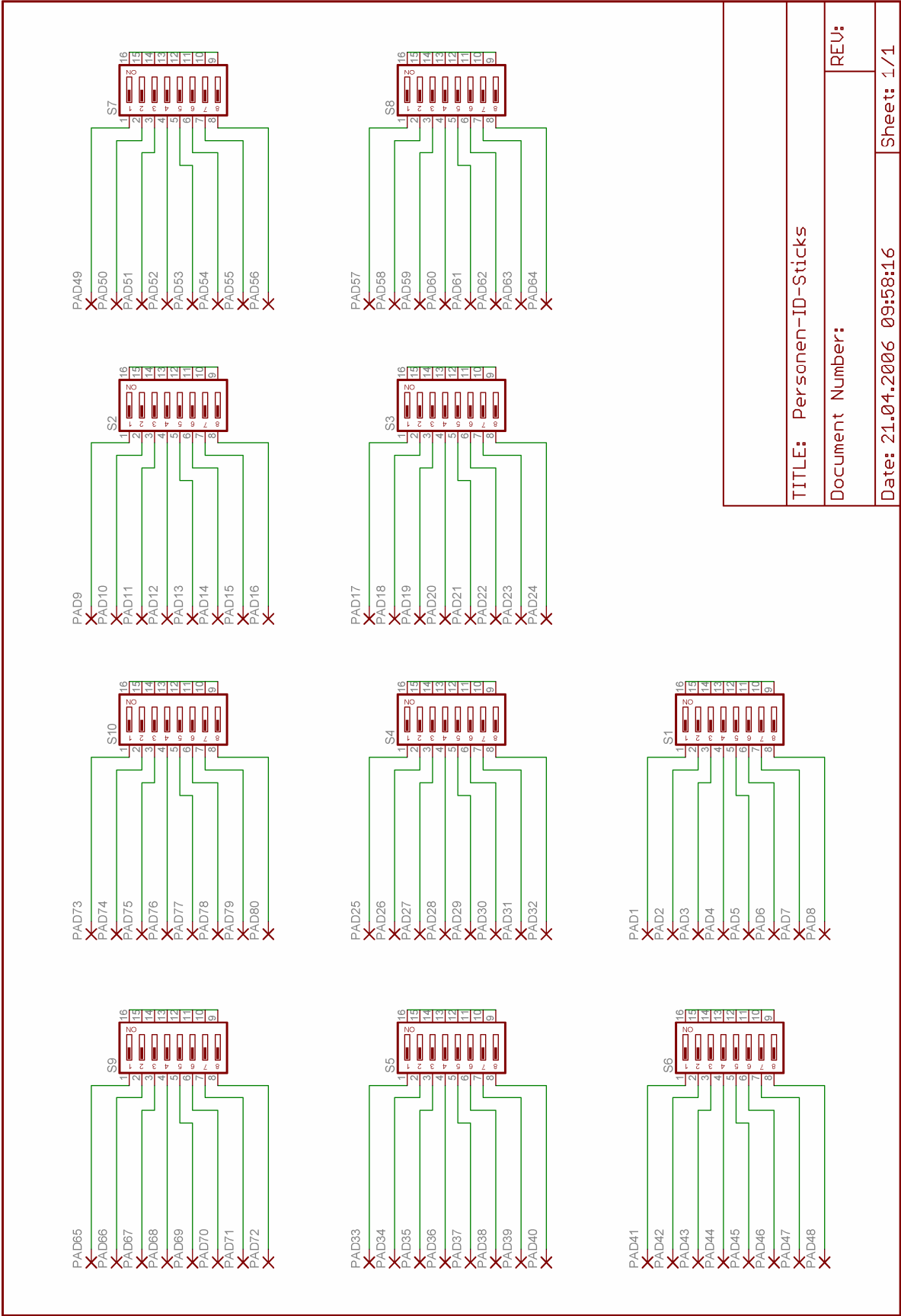
Unsere erste Überlegung zur Realisierung eines ID-Schlüssels wurde bereits in der Projektvorüberlegung besprochen und sah einen analogen PID-Stick vor, der jedoch durch die Einfachheit eines digitalen Codes überstimmt wurde. Somit war die Idee des Achtbit-Code-Schlüssels geboren. Jedoch traten dabei schon die ersten Probleme auf, denn wir brauchten einen möglichst kleinen, robusten und schnell zu verriegelnden Stecker der mindestens acht Steckkontakte beinhaltet. Ideal wäre ein Stecker im bekannten USB-Format mit acht Kontakten und industriellem Metallgehäuse. Da die meisten am Markt befindlichen Stecker mit dieser Anzahl von Kontakten aber einfach zu groß und zu unhandlich sind, entschieden wir uns für die naheliegendste Lösung, den Modularstecker RJ45. Dieser besitzt acht Kontakte, ist einfach zu bedienen und sicher zu verriegeln. Um jedoch die Codes mit Hilfe von Dipschaltern auf ein bestimmtes Potential einzustellen, benötigen wir einen Spannungsbezugspunkt. Dazu definierten wir alle ID-Codes auf Low-Aktiv, d.h. die Kodierungsbits werden im Aktiv-Status mit dem Schalter auf Masse und alle anderen Datenleitungen durch die direkt am PIO-Port liegenden Pull-Up-Widerstände auf High-Level „gezogen“. Allerdings muss ein Schalter immer die Masseleitung als Bezugspotenzial durchschalten, so dass wir insgesamt nur noch sieben Schalter für die Kodierung zur Verfügung haben. Durch das Masse-bezugspotenzial beugen wir zudem eventuellen Kurzschlüssen an der offenen Modular RJ45 Buchse vor, da keine Spannung an der Buchse übertragen wird.

4.5.2 Aufbaubeschreibung

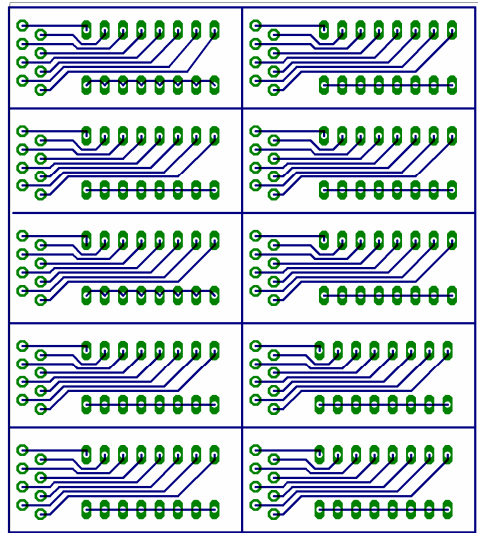
Die Personen-ID-Sticks sind recht einfach aufgebaut. Mittels eines achtfachen Dipschalters wird, je nach Stellung von Schalter 1-7, eine individuelle persönliche ID erstellt. Schalter acht muss immer in Stellung „ON“ sein, da dieser mit der Masseleitung verbunden ist und das achte Bit intern an Masse liegt. Dadurch ergibt sich ein Siebenbit-Codewort, also insgesamt maximal 2^7 (128) verschiedene ID's die an dieses System angeschlossen werden können. Der Dipschalter ist auf einer kleinen Platine befestigt und diese ist wiederum über ein kurzes Kabel mit einem achtfachen Modularstecker (RJ45) verbunden, welcher extern an die PID-IN-Platine gesteckt wird.



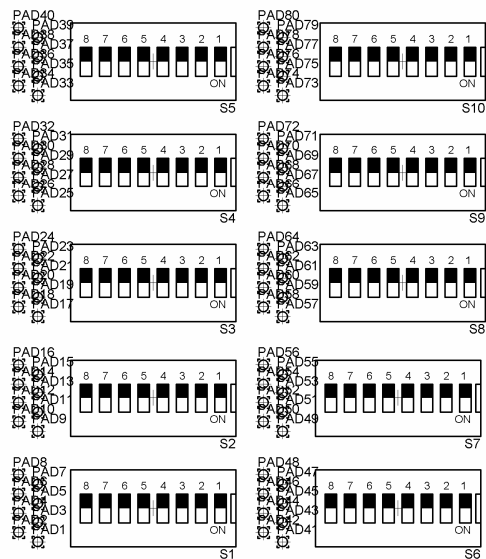
4.5.3 Schaltplan PID-Sticks



4.5.4 Layout PID-Sticks



4.5.5 Bestückungsplan PID-Sticks



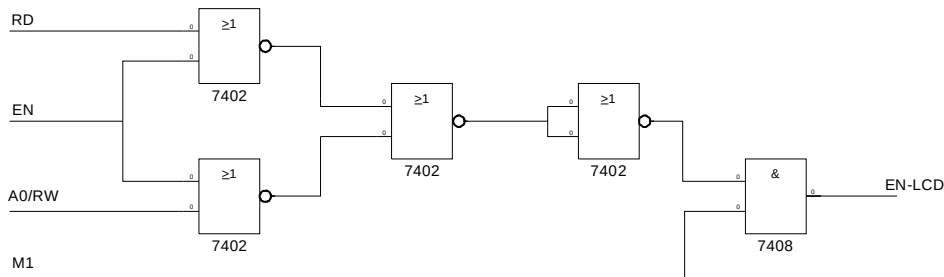
4.6 DISTAS (Display- und Taster-Platine)

4.6.1 Konstruktionsüberlegungen

Die Benutzerschnittstelle zwischen Gerät und Bediener sollte möglichst einfach und kompakt auf der Frontplatte des Gehäuses verwirklicht werden. Dementsprechend sollten alle Bedienelemente wie Taster und Leuchtdioden in der Nähe des Displays angeordnet sein. Alle drei Displays werden übersichtlich nebeneinander angeordnet und können somit auch „gleichzeitig“ bedient werden. Da fast alle Taster ein gewisses Prellverhalten haben, welches für die digitale Signalverarbeitung letal sein kann, hatten wir uns eine einfache Entprellschaltung überlegt, die jedoch nicht das erzielte Ergebnis lieferte. Somit wurde die Entprellschaltung vorerst vernachlässigt, da das Verhalten der digitalen Eingänge mit oder ohne Entprellschaltung offensichtlich nicht relevant zu sein schien. Zudem haben wir sehr prellarme Taster verwendet, die bereits eine sehr geringe Prellzeit von max. 100µs aufweisen.

Damit wir von dem Z80-System möglichst einfach Zeichen auf einem Display darstellen können, wählten wir ein Display das bereits einen integrierten ASCII-Controller „onboard“ hat und somit kein Grafikdisplay für eine ansprechendere Optik. Der Vorteil dieser Displays liegt darin, dass man nur den entsprechenden Hex-Code eines ASCII-Zeichens als Datenwort auf den Datenbus geben muss. Um das Display auch bei schlechten Sichtverhältnissen gut lesen zu können, wählten wir ein einzeliliges Display mit möglichst großen Zeichen, welches zusätzlich durch eine Hintergrundbeleuchtung illuminiert werden kann.

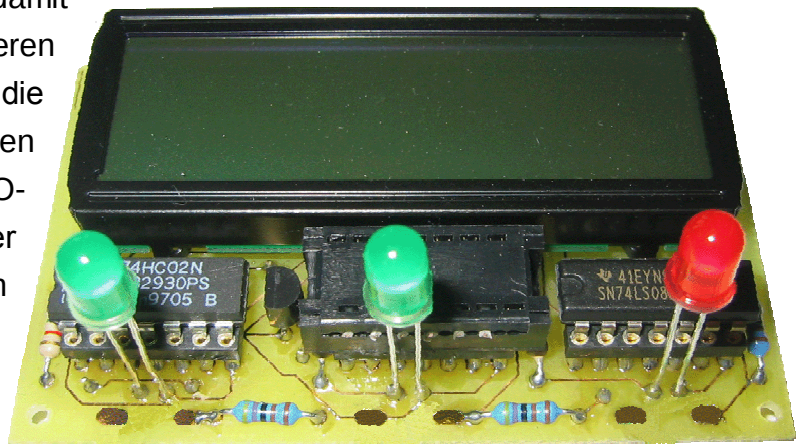
Um die Displays anzusteuern benötigt man das Chipenable-Signal CE. Die Displays besitzen Readregister, womit man den Busy-Status oder auch den angezeigten Displayinhalt auslesen kann. Eine fehlerhafte Software könnte nun bei einem einfachen Display-Interface zur Zerstörung der Hardware führen, indem versucht wird mit Hilfe des Ausgabebefehls OUT ein Readregister des Displays anzusteuern. In diesem Falle würden beide Busteilnehmer auf dieselben Busleitungen schreiben wollen und es würde zu einer Kollision kommen. Um diesen möglichen Softwarefehler auszuschließen gibt es eine hardwareseitige Zusatzlogik, die die Freigabe des Displays mit Hilfe der Auswertung des Read-Signals RD bei einer Ausgabe auf das Readregister der Displays verhindert.



Displayzusatzlogik

4.6.2 Aufbaubeschreibung

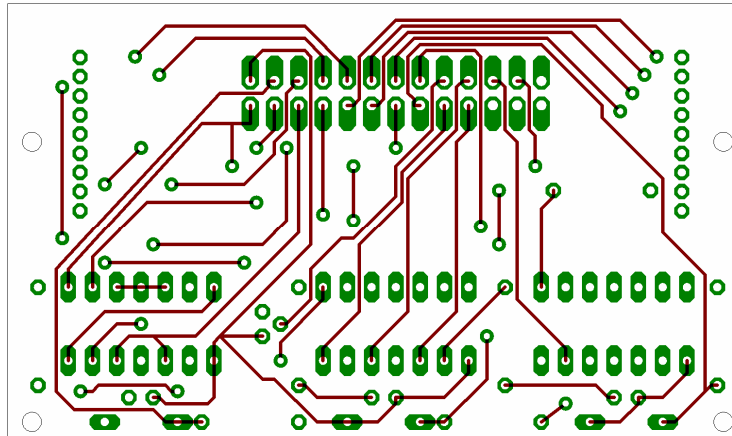
Mit einer minimalen Anzahl von drei Bedientastern pro Displayeinheit erreichten wir einen maximalen Bedienkomfort, da wir mit den Tasten (auf, ab, OK) das komplette Displaymenü bedienen können. Die LED's sind direkt unter dem Display auf der gleichen Platine untergebracht, damit sie den Status auffällig signalisieren können. Ursprünglich sollten die LED's aufgrund des begrenzten Ausgangsstroms der PIO-Ausgänge von max. 4mA über einen Treiberbaustein angesteuert werden. Da wir aber Low-Current-LED's mit einer Stromaufnahme von 2mA



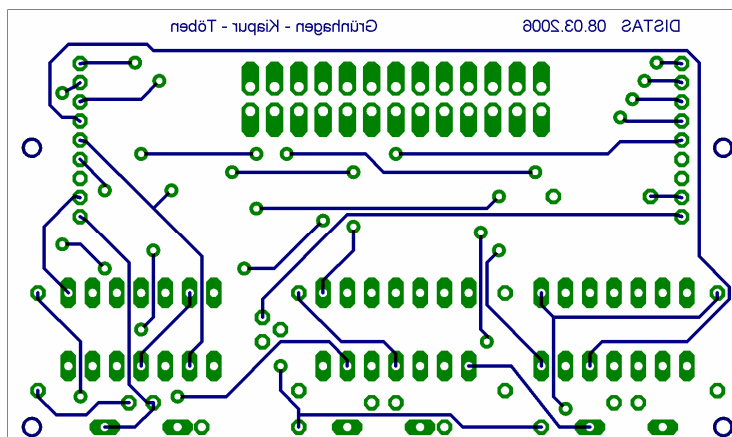
verwendet haben, wurde der Treiberbaustein überflüssig und durch einen IC-Sockel mit 1:1 Verdrahtung ersetzt. Die LED's sind jeweils mit einem Vorwiderstand beschaltet, der sich wie folgt berechnet: High-Aktiv Ausgang der PIO führt 5V, die LED's benötigen 2V mit 2mA, somit ergibt sich ein Vorwiderstand von $R = \frac{3V}{2mA} = 1500\Omega$

Gleiches gilt für die Hintergrundbeleuchtung der Displays, die über einen Vorwiderstand von 90 Ohm an einem Feldeffekt-Transistor BS170 als Treiber angesteuert wird. Die oben beschriebene Displayzusatzlogik zur Freigabe des Displays wurde durch zwei Logik-IC's direkt auf der Platine realisiert. Aufgrund der möglichst geringen Gehäusemaße, sollte die Display-Platine sehr kompakt auf einer doppelseitigen Platine entwickelt werden. Somit ergab sich, dass der Hauptanschlussstecker auf der entgegengesetzten Seite der Bestückungsseite liegt. Die low-aktiven Taster benötigen je einen Pull-Up-Widerstand für den definierten Zustand im Normalbetrieb, welche ebenfalls direkt neben den Tasteranschlüssen auf der Displayplatine vorhanden sind.

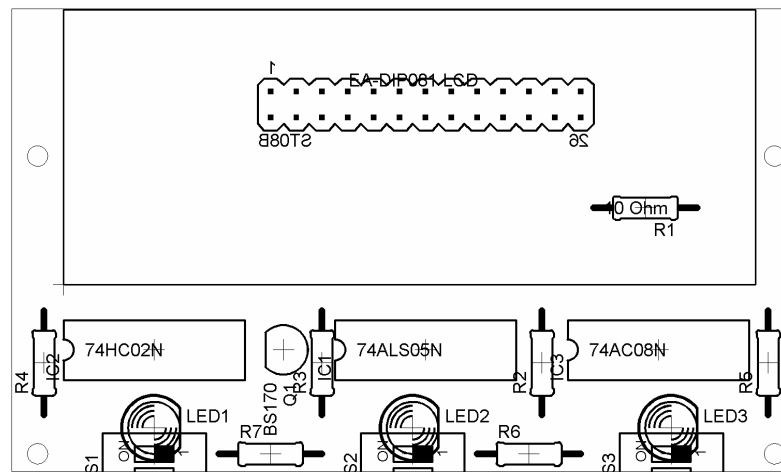
4.6.4 Layout DISTAS (Oberseite)



4.6.5 Layout DISTAS (Unterseite)



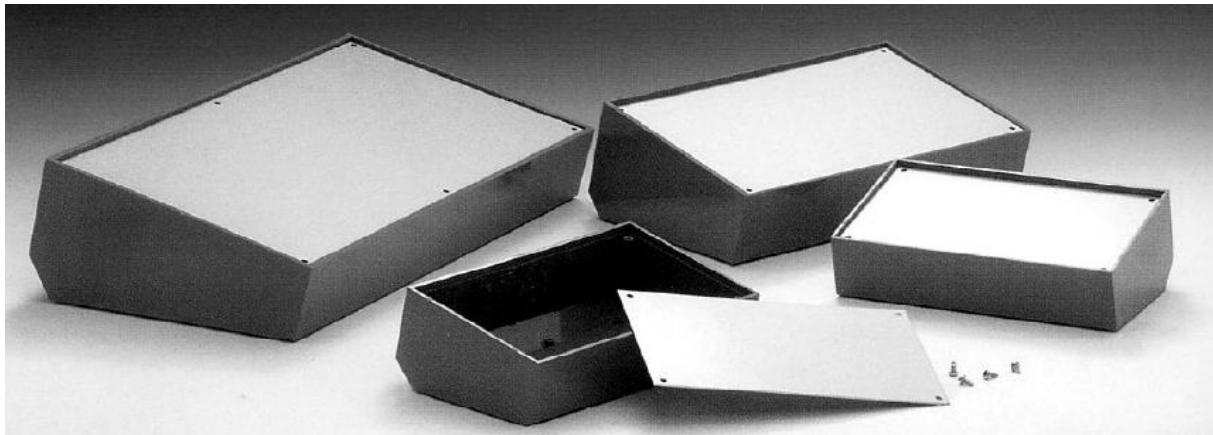
4.6.6 Bestückungsplan DISTAS



4.7 Gehäuse

4.7.1 Konstruktionsüberlegungen

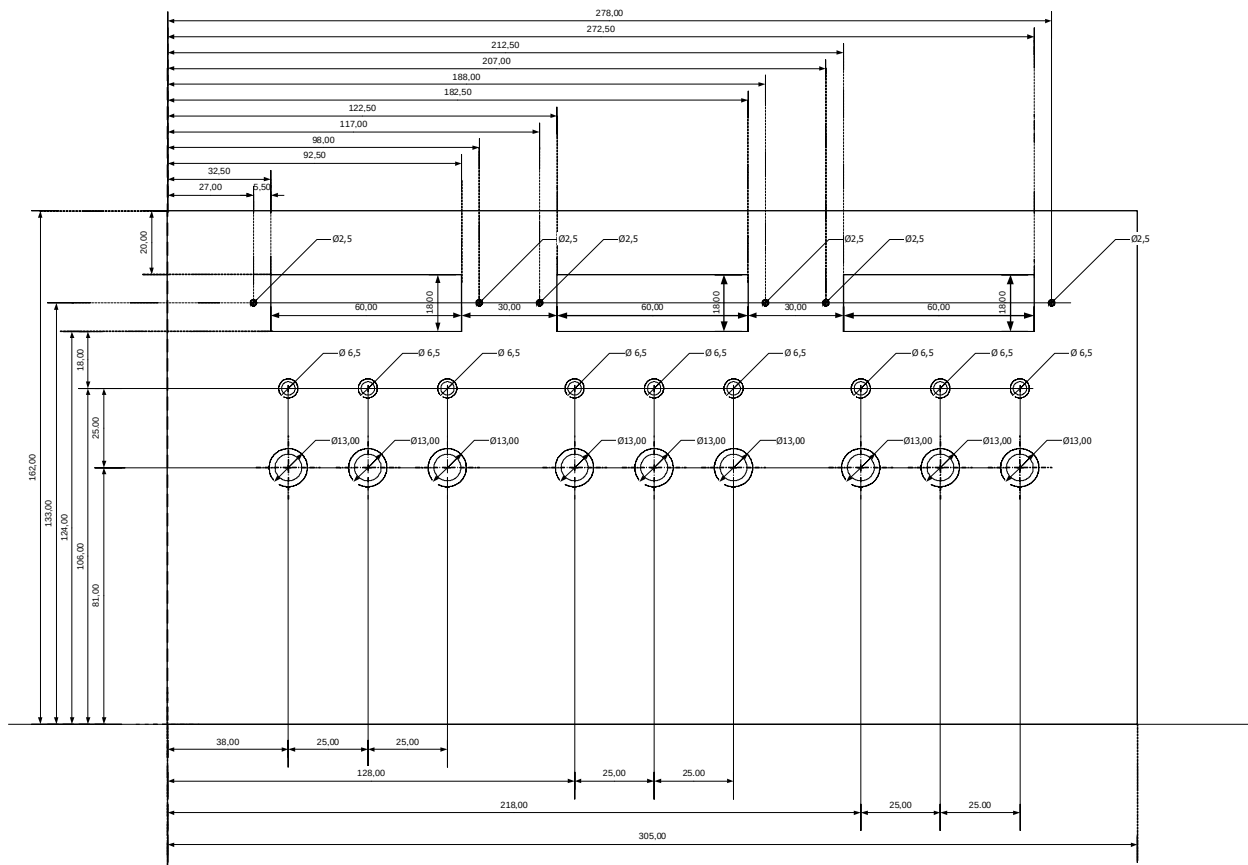
Für die gute Handhabung des Gerätes während eines Einsatzes ist ein möglichst kleines, handliches und portables Gehäuse sinnvoll. Zusätzlich sollte es für den „Outdooreinsatz“ stoßfest und Spritzwassergeschützt nach Schutzart IP 65 sein. Dementsprechend wäre ein ABS-Plastik Gehäuse mit Schutzart IP65 von der Firma BOPLA möglich, jedoch sind die Komponenten dieses Systems etwas größer als geplant ausgefallen, so dass das ausgewählte BOPLA-Gehäuse bereits zu klein wurde. Somit fiel die Entscheidung auf ein Pultgehäuse von Teko, das etwas breiter als das BOPLA-Gehäuse ist und zusätzlich eine Pultschräge zur besseren Bedienung bietet. Jedoch ist es nicht spritzwassergeschützt und stoßfest und somit als Prototyp nur begrenzt im Außenbereich einsetzbar.



4.7.2 Aufbaubeschreibung

Für die diversen Bedienelemente und Schnittstellen musste das Gehäuse individuell angepasst werden. Für die Displays, Taster und die LED's wurden die Aussparungen in die Aluminiumfrontplatte gefräst, sowie die rechteckigen Aussparungen für die PID-Stick-Schnittstellen an der vorderen Längsseite. Die zusätzlichen Bohrungen für die serielle Schnittstelle, Lichttaster, Hauptschalter und DC-Buchse für die Ladespannung wurden manuell an den Stirnseiten gefertigt. Zur Beschriftung der Elemente wurde eine spezielle Klebefolie bedruckt und auf die Frontplatte aufgeklebt. Die Platinen wurden auf eine zusätzlich eingebaute Bodenplatte montiert, wobei einige davon aus Platzgründen übereinander montiert wurden.

4.7.3 Frontplatte



Trupp 1

↑ ↓ OK

Trupp 2

↑ ↓ OK

Trupp 3

↑ ↓ OK

ASG-Überwachung TGK-3000

5 Software

5.1 Softwarestruktur

Die im Gerät verwendete und zuvor beschriebene Hardware benötigt nun also noch eine Software um zu arbeiten. Dieser so genannte „Quellcode“ muss von der Assemblersprache in den eigentlichen OP- oder Maschinencode kompiliert werden, um wiederum von der CPU verstanden zu werden.

Da wir jedoch von vornherein davon ausgingen, dass das Betriebssystem eine umfangreiche Programmierung mit sich bringen wird, haben wir uns dazu entschieden, den Großteil der Programme in der Programmiersprache ‚C‘ zu realisieren. Sämtliche Abfragen, Verzweigungen usw. lassen sich dadurch leichter und übersichtlicher programmieren.

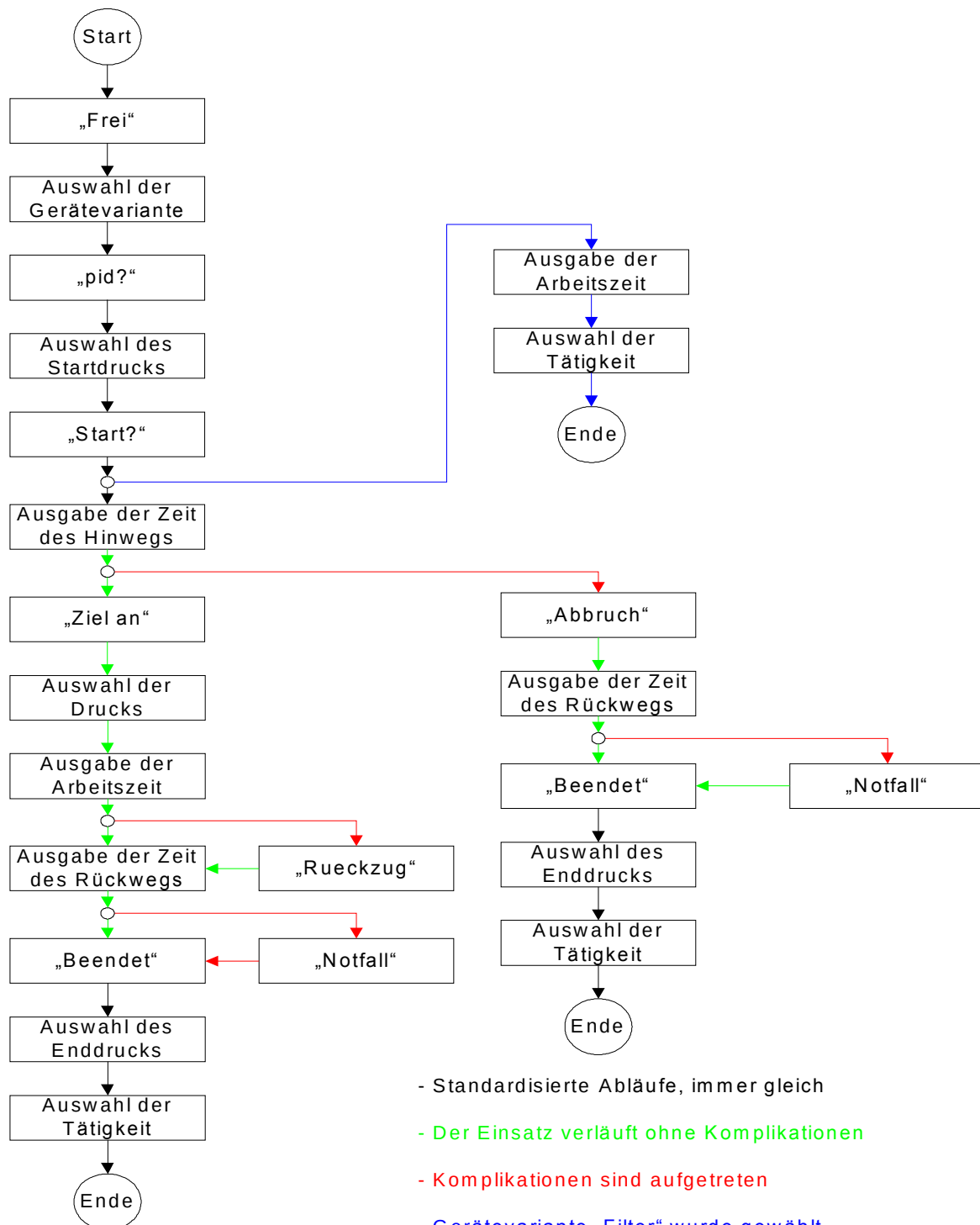
Somit müssen die von uns geschriebenen Programme jedoch zunächst in einen Assemblercode gewandelt und anschließend mit einem Linker zusammengefügt und ausführbar gemacht werden. So wird z.B. aus unserem Hauptprogramm `tgk3000.c` vorerst die Datei `tgk3000.asm` und dann verlinkt mit einigen externen Modulen, zur für das System ausführbaren Datei `tgk3000.z80`. Diese kann dann mit einem Programmiergerät für EPROM's auf den verwendeten ROM unseres Systems „gebrannt“ werden.

Diverse Programmteile, welche im Folgenden näher beschrieben werden, hätten durchaus simpler oder anders gestaltet werden können. Darauf haben wir allerdings bewusst verzichtet, um evtl. Programmänderungen leichter zu ermöglichen. Diese Änderungen werden sich erst in der realen Testphase des Gerätes aufzeigen, da viele Feuerwehren eigene Standards gesetzt haben.

Die gesamte Software ist unterteilt. Zum einen haben wir den eigentlichen Aufruf des Hauptprogramms durch ‚`akemain()`‘, dies ist für den Betrieb eines solchen Programms auf einer CPU unbedingt erforderlich. Des Weiteren existiert natürlich das Hauptprogramm, aus welchem heraus Initialisierungsschleifen sowie sämtliche Unterprogramme aufgerufen werden. Diverse Programmteile inkludieren so genannte Systemmodule, mit welchen z.B. Ein- und Ausgaben zu einzelnen Ports standardisiert und realisiert werden.

5.1.1 Visuelle Struktur der Software

Um den Ablauf der Software zu veranschaulichen sei hier einmal ein Programmablauf mit allen möglichen Verzweigungen aufgezeigt.



In den mit „Auswahl“ gekennzeichneten Menüfeldern stehen folgende Möglichkeiten zur Verfügung:

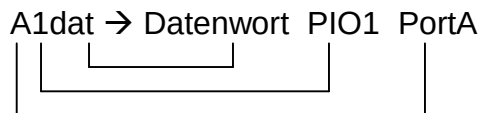
Menüpunkt	Möglichkeiten
Gerätevariante	„PA 300“ „ChemieSA“ „Hitzesch“ „Regen PA“ „Filter“
Behälterdruck vor Einsatz	„270“ bis „320“ Bar
Behälterdruck bei „Ziel an“	„0“ bis „300“ Bar
Behälterdruck bei Einsatzende	„0“ bis „300“ Bar
Tätigkeit	„Retten“ „Loeschen“ „Erkunden“ „Dichten“ „Auffange“ „Umpumpen“ „Funkverb“

5.2 Headerdatei „io.h“

Aufgrund der Hardwarekomplexität und der damit verbundenen Vielzahl von Peripherieadressen ist es von Vorteil, eine Headerdatei anzulegen.

In dieser Datei werden Makros für die gesamte Peripherie definiert, welche von allen Unterfunktionen verwendet werden können. Somit kann übersichtlicher programmiert werden, da zur Ansteuerung sämtlicher Geräte nun keine hexadezimalen Adressen mehr benötigt werden, sondern hierfür verwendete Makros.

Die Makronamen sind von den Geräten und Verwendungszwecken abgeleitet und somit in jedem Unterprogramm zuzuordnen. Somit wird z.B. aus der „nichts aussagenden“ Adresse 0x00 das Makro `a1dat`.



Neben den Hardwareadressen werden einige Systemmodule ebenfalls durch Makros ersetzt. Auch diese sind somit leicht mit ihrer eigentlichen Funktion in Verbindung zu bringen.

5.3 Systemmodule

5.3.1 Systemmodul `,delay()`

Diese Funktion ist eine universelle Zeitschleife. Ihre Basis ist auf Millisekunden bemessen. Der gewünschte Wert wird dem Programm als Konstruktor zugeteilt. Mit dem Befehl „`delay(2000)`“ wird zum Beispiel eine Verzögerung von 2000ms, also 2sec erzielt.

5.3.2 Systemmodul `,busy1()`

Dieses Modul dient der Kommunikation mit den Displaymodulen. Soll sowohl ein Datenwort, als auch ein Steuerwort an eines gesendet werden, so muss sichergestellt werden, dass der Displaycontroller empfangsbereit ist und alle vorherigen Befehle erfolgreich abgearbeitet hat.

Für jedes Display ist eine Busy-Schleife inkludiert. Der Aufruf der Module erfolgt über den „`call`“-Befehl aus allen Funktionen, die mit dem Display kommunizieren.

5.3.3 Systemmodul ,output()'

Zum Übermitteln verschiedenster Daten an beliebige Hardwareadressen kann diese Funktion verwendet werden. Der Aufruf erfolgt über „output(X, Y)“. In Y stehen die zu übermittelnden Daten, welche an die Adresse X gesendet werden sollen.

5.3.4 Systemmodul ,in()'

Hiermit ist es möglich, den Inhalt verschiedenster Peripheriegeräte einzulesen und in eine beliebige Variable einzutragen. Mit dem Befehl „X= input(Y)“ wird also der Zustand der Adresse Y unmittelbar in die Variable X eingetragen.

5.3.5 Systemmodul ,put1()'

Wird ein einzelnes Zeichen vom Typ „char“ zum Display übertragen, so wird dieses Modul verwendet. Aufgerufen wird es aus einer beliebigen C-Routine mit dem Befehl „put1(X)“. In diesem Fall würde der Inhalt von X an das Display 1 gesendet. Diese Funktion ist aufgrund der Kommunikation mit den Displays auf das Modul ,busy1()' angewiesen.

5.3.6 Systemmodul ,outstr1()'

Will man Zeichenketten zum Display übertragen, ist dies nicht ohne weiteres möglich. Dieses Modul zerlegt visuell die Zeichenkette in einzelne Zeichen und übermittelt diese dann nacheinander mit Hilfe des Moduls ,put1()' an das Display. Da wir ein ASCII-taugliches Display verwendeten, werden auch die auf dem Display vermeintlichen Zahlen als reine Zeichenkette, also als „String's“ übermittelt.

5.3.7 Systemmodul ,lcd1()'

Zum Initialisieren der Displays wird diese Routine verwendet. Man ist dadurch in der Lage Steuerwörter, die an ein Display gesendet werden sollen, mit einem einfachen Befehl an das Control-Register der Anzeige zu senden.

5.4 Startmodul ,akemain()'

Diese Funktion dient der Vorbereitung für alle anderen kommenden Module. Um mit dem Stackpointer arbeiten zu können, muss dieser zunächst auf eine Adresse im Stack gesetzt werden. Aus der Speicherplatzadressierung in Kapitel 4.1.6.2 ergibt sich ein Adressenbereich von 8000 bis A000 für den RAM. Da der Stack ein LIFO-Speicher (Last In First Out) ist, muss die Stackpointeradresse im oberen RAM-Bereich sein. Daraus lässt sich die Adresse 8EFF ableiten.

Nachdem der Stackpointer gesetzt ist, kann nun das Hauptprogramm ,main()' aufgerufen werden.

Ist das Programm ,main()' beendet, folgt der Programmaufruf „halt“ und das System bleibt an dieser Adresse stehen.

5.5 Hauptprogramm ,main()'

5.5.1 Programmbeschreibung

Bei dieser Funktion handelt es sich um das Hauptprogramm.

Aufgrund der Bedingung, drei Trupps gleichzeitig und vor allem unabhängig voneinander überwachen zu können, ist dieses Programm sinnbildlich dreigeteilt.

Zunächst werden bei Inbetriebnahme des Systems durch Aufruf von ,main()' die Initialisierungsschleifen ,iniim2()' sowie ,init()' ausgeführt. Im Anschluss daran wird das Auslösen von Interrupts an der CPU freigeschaltet. Dies wird in ,iniim2()' lediglich vorbereitet. Nun wird der Interruptmode 2 durch den Befehl „im 2“ am System eingerichtet. Zu Port A und B der PIO 2 wird das Steuerwort 83_n übermittelt. Dadurch wird auch die PIO für Interruptverarbeitung freigeschaltet.

Folgend werden sämtliche Variablen, welche in diversen Unterfunktionen verwendet werden, initialisiert.

An diesem Punkt beginnt die For-Schleife, welche unendlich immer wieder durchlaufen wird.

Es wird zu Beginn dieser Schleife kontrolliert, ob die Variable ,enable1' (im Folgenden auch Schleifenfreigabe genannt) gesetzt wurde. Trifft dies zu, wird unmittelbar der Schleifenzähler des ersten Trupps namens ,trupp1' inkrementiert und die Schleifenfreigabe wird unverzüglich zurückgesetzt. Die Schleifeninternen Variablen ,sz1, sz11, sz12, sz13' sowie ,rechnen1' werden ebenfalls zurückgesetzt.

Dies muss gemacht werden, da sie in diversen Unterfunktionen für andere Zwecke verwendet werden.

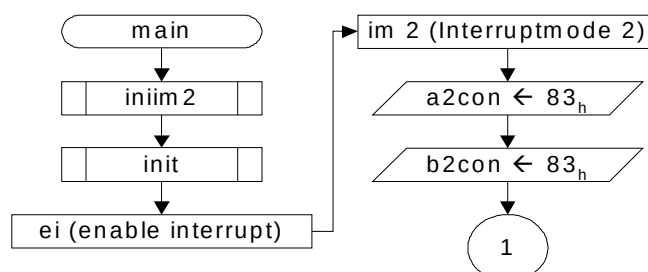
Im Anschluss an die Kontrolle von ‚enable1‘ wird überprüft, ob der Schleifenzähler ‚trupp1‘ den Inhalt 11 hat. Ist dies so, wird ‚p1‘ inkrementiert. In ihr wird die Einsatznummer während des Betriebs, fortlaufend gezählt. Alle LED's des 1. Truppmoduls werden ausgeschaltet und somit sämtliche Variablen, die im Programm durch Abarbeiten des vorherigen Einsatzes verändert wurden. Nun folgt der eigentliche Einstieg, in dem diverse Unterprogramme bzw. Aktionen aufgerufen werden, um die Überwachung eines Trupps zu ermöglichen. Hierfür wird mit Hilfe einer Switch-Abfrage der Inhalt der Variable ‚trupp1‘ überprüft und dann dementsprechend reagiert. Nebestehende Reaktionen sind möglich.

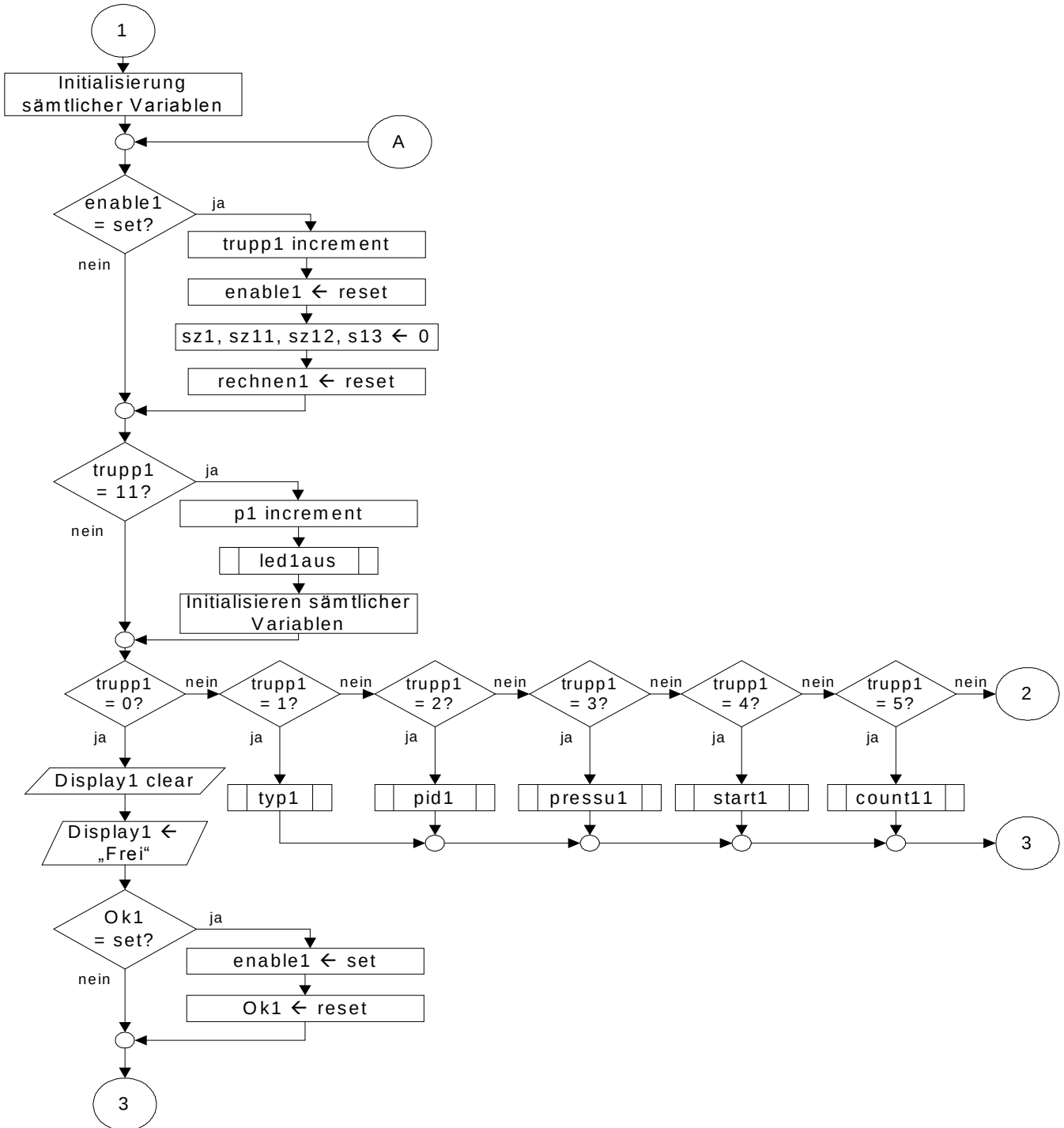
‚trupp1‘	Aktion
0	Ausgabe „Frei“
1	typ1()
2	pid1()
3	pressu1()
4	start1()
5	count11()
6	ziel1()
7	count12()
8	count13()
9	ende1()
10	art1()

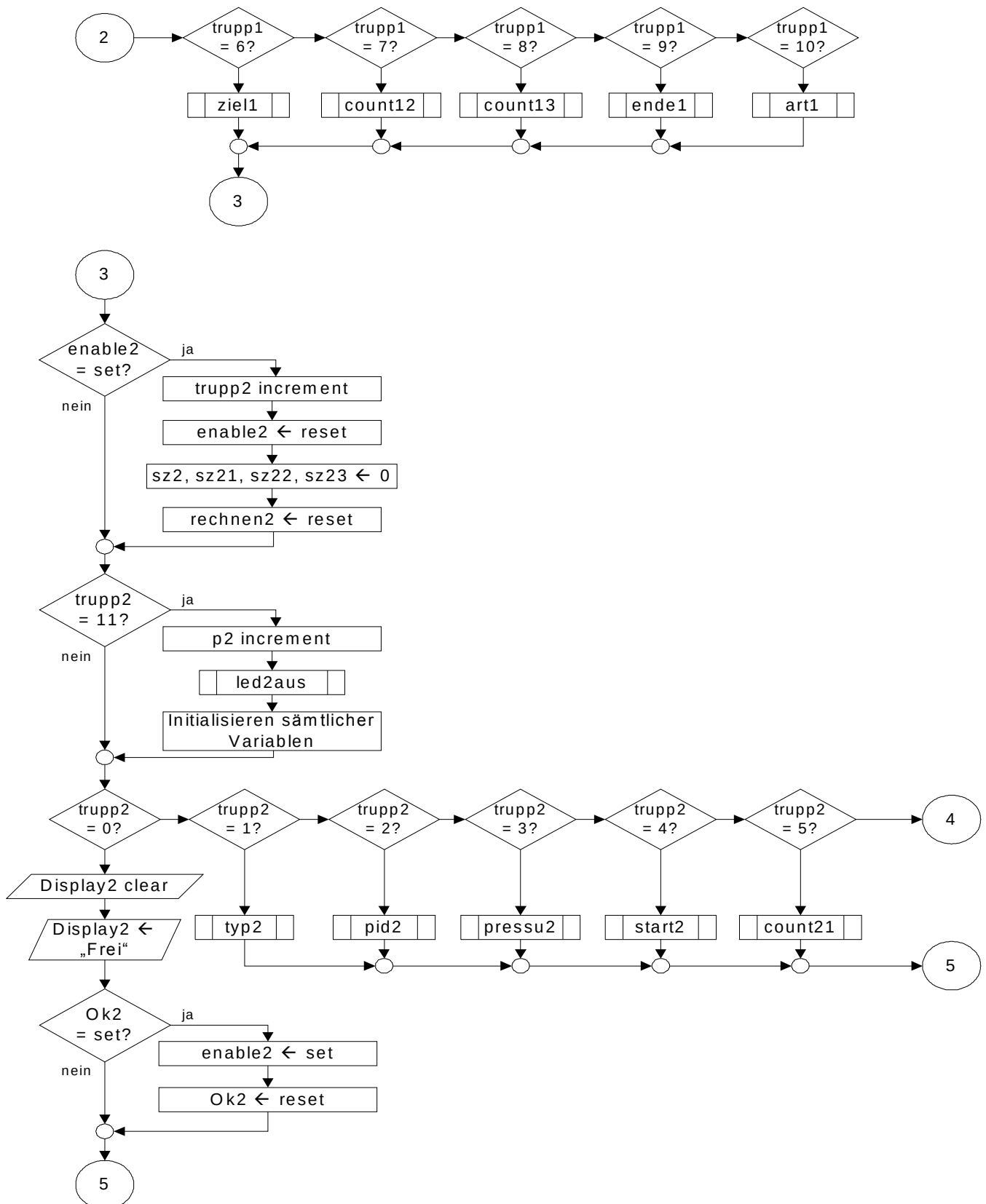
Hiermit ist die Bearbeitungsroutine für den ersten Trupp abgeschlossen. Diese Prozedur wird nun für die beiden übrigen Truppmodule wiederholt. Lediglich die Variablen bzw. Unterprogrammnamen ändern sich. Ist die Bearbeitung aller drei Trupps abgeschlossen, erfolgt noch der Aufruf des Zeitmoduls ‚delay()‘ um eine Verzögerung von 200ms zu erreichen, sowie das Modul ‚extras()‘ wird ausgeführt. Hiernach wird, bedingt durch die For-Schleife zum Beginn der ‚trupp‘-Abfrage gesprungen.

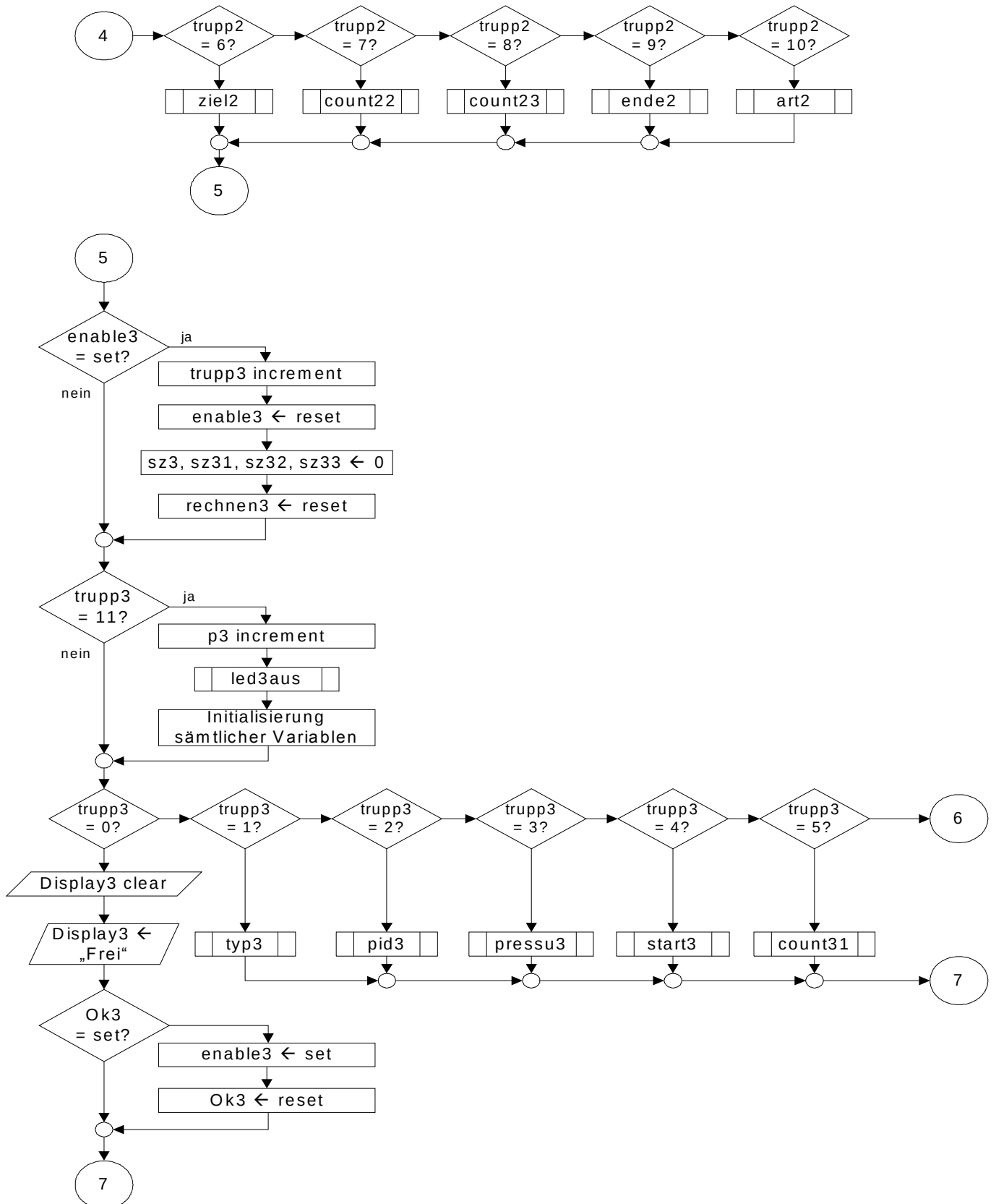
In den folgenden Beschreibungen werden die Module, die je Trupp aufgerufen werden und somit dreifach vorhanden sind, lediglich einmal beschrieben. Markante Unterschiede der truppspezifischen Module werden erläutert. Auf Variablen, deren Namen sich lediglich ändern, wird nicht näher eingegangen, da die Zugehörigkeit zu den Trupps am Namen ersichtlich wird.

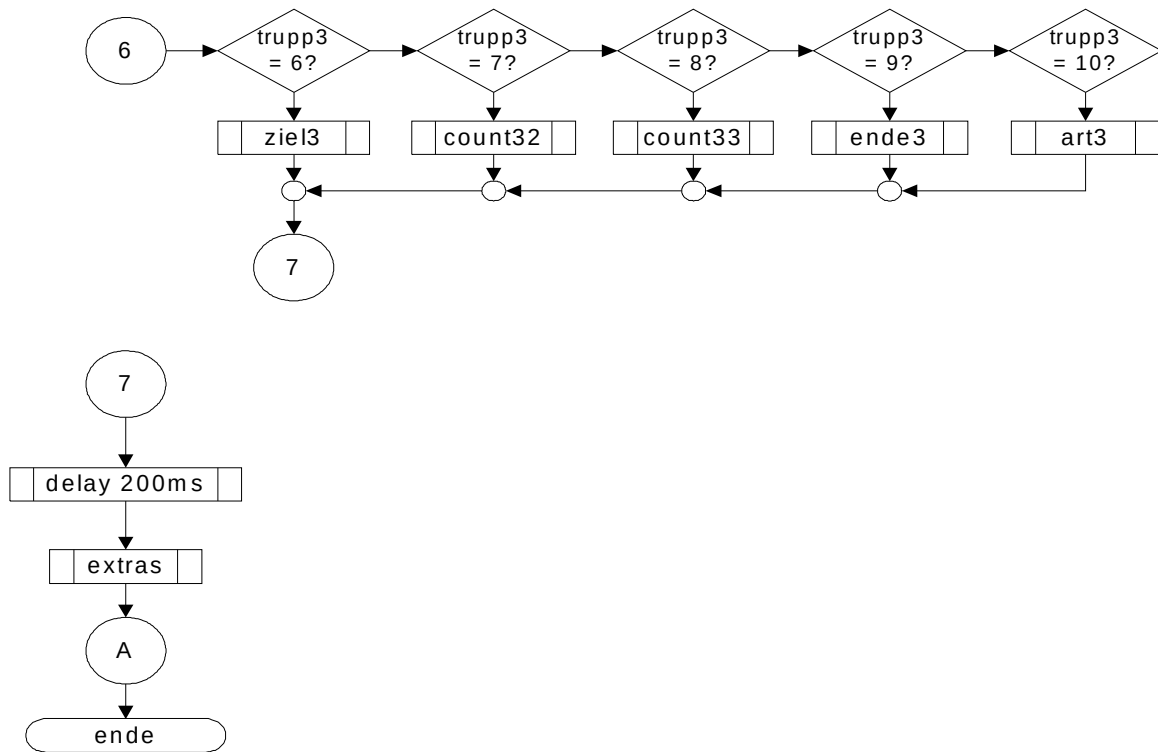
5.5.2 Programmablaufplan ‚main()‘











5.6 Interruptroutinen

5.6.1 Programmbeschreibung irouta() und iroutb()

Sobald einer der Taster auf der Frontplatte betätigt wird, löst dieses einen Interrupt aus. Je nachdem, an welchem Port der PIO 2 dieser Aufruf erfolgt, wird in die entsprechende Vektortabelle verwiesen. Hierin erfolgt der Aufruf von ,irouta()' bei Betätigung eines Tasters an Port A und bei Auslösen über Port B wird ,iroutb()' angesprochen.

Zunächst wird hier näher auf ,irouta()' eingegangen.

Die Routine startet mit einer Verzögerung von 50ms durch den Aufruf von ,delay()'. Dies dient der Softwareentprellung der Taster (Siehe Kapitel 6.2).

Danach wird der Zustand von Port A der PIO 2 eingelesen und in der Variable ,in1' abgespeichert. Diese Daten werden nun an ,in2' übergeben, anschließend mit DB_h maskiert und in die Variable ,ina' geschrieben. Dies ist erforderlich, da die PIO im Bitcontrol-Modus betrieben werden. Somit können die als Ausgang verwendeten Bits einen Zustand haben, der das Ergebnis folgender Tasterselektierung verfälschen würde. Ist die Übergabe geschehen, folgt eine Switch-Abfrage der Variablen ,ina'. Je nach Zustand wird nun der zugehörige Merker gesetzt. Die Abhängigkeiten zwischen den Tastern und Merkern ist in der Tabelle aufgeführt.

Es folgt eine erneute Verzögerung durch das Programm ,delay()'.
Da durch einen Interrupt das gesamte System in Stillstand versetzt wird, muss es nach der Verarbeitung folglich auch wieder in Betrieb gesetzt und das Auslösen neuer Interrupts ermöglicht werden. Dies geschieht durch den Befehl „ei“ (enable interrupt), welcher unmittelbar zur CPU gesendet werden muss.

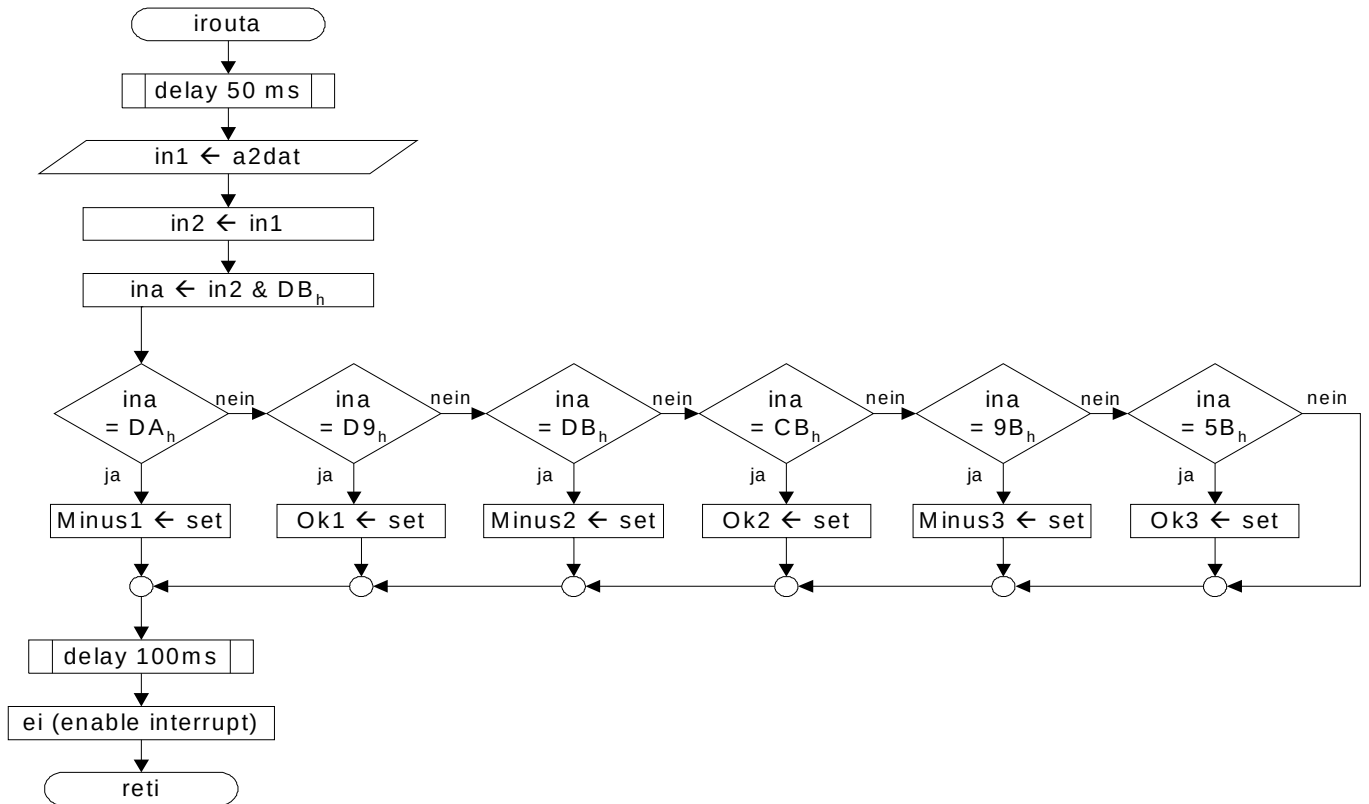
Taster	Port	Bit	Maske (Binär)	Maske (Hex)
Ok1	A	0	11011010	DA
Plus1	B	0	01001000	48
Minus1	A	1	11011001	D9
Ok2	A	3	11010011	D3
Plus2	B	3	01000001	41
Minus2	A	4	11001011	CB
Ok3	A	6	10011011	9B
Plus3	B	6	00001001	09
Minus3	A	7	01011011	5B

Im Gegensatz zu anderen

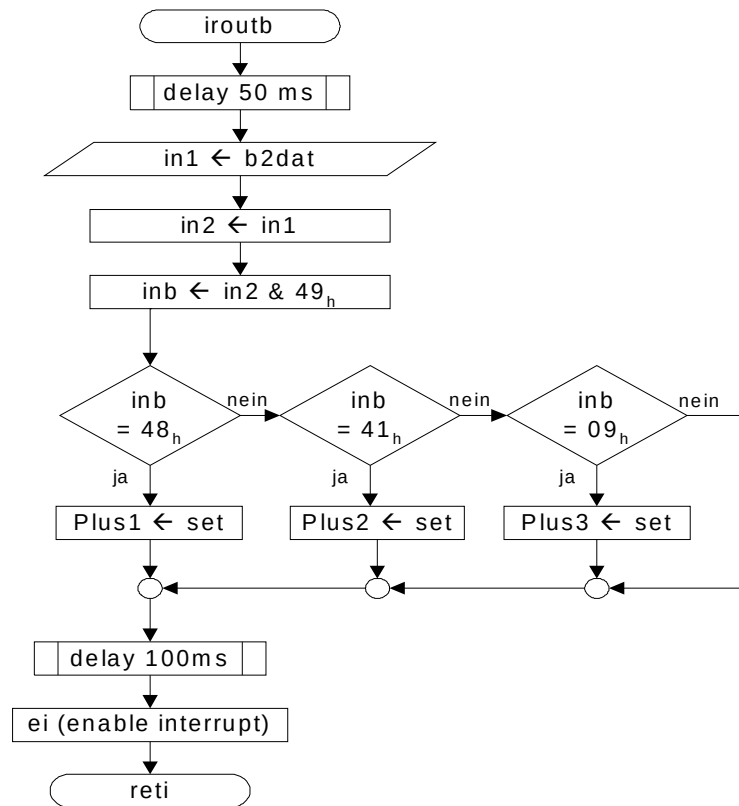
Modulen wird dieses Modul mit dem Befehl „reti“ verlassen, um einen direkten Rücksprung in das zuvor ausgeführte Modul zu ermöglichen.

Die Routine ,iroutb()' ist identisch zu ,irouta()' aufgebaut. Der Unterschied liegt lediglich in den Namen der Variablen sowie in den Abhängigkeiten zwischen den Tastern und Merkern. Diese Werte sind ebenso in der o.a. Tabelle aufgeführt.

5.6.2 Programmablaufplan ,irouta()'



5.6.3 Programmablaufplan ,iroutb()'

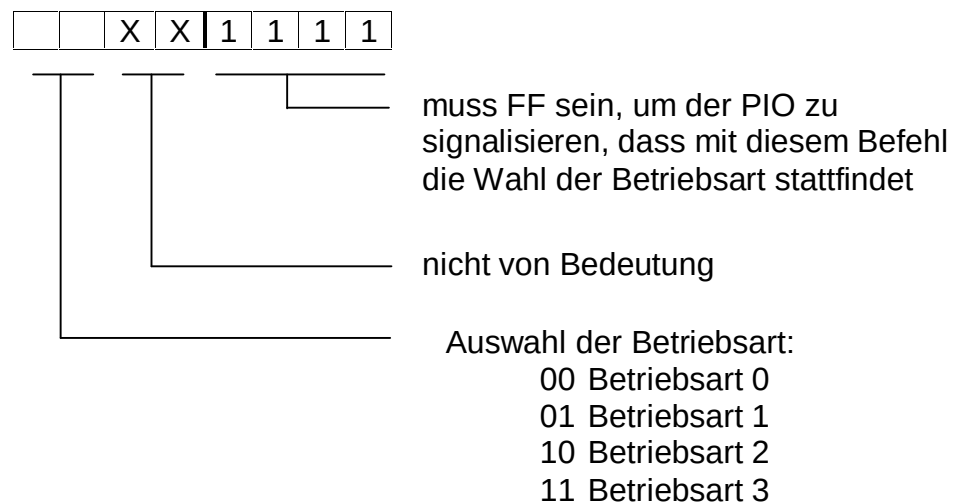


5.7 Unterprogramme

5.7.1 init()

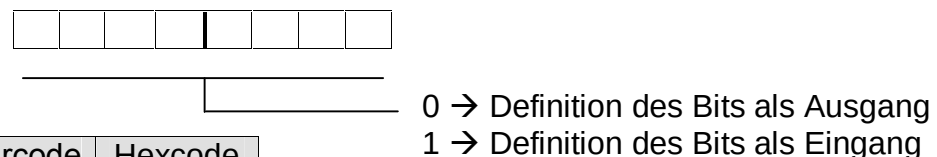
5.7.1.1 Programmbeschreibung

Diese Routine wird lediglich beim Einschalten des Systems ausgeführt. Dies ist erforderlich, um sämtliche Peripherie-Geräte zu konfigurieren. Wie bereits bei der Aufbaubeschreibung der PIO-Verteiler-Platine erwähnt, betreiben wir die PIO's in Mode 3. Man spricht auch vom Bitcontrolmode. Das Steuerwort, um diese Betriebsart einzurichten, ergibt sich wie folgt:



Hat man dieses Steuerwort mit Hilfe der Funktion ,output()' an die Control-Adresse der PIO übermittelt, erwartet sie ein unmittelbar folgendes Steuerwort, mit dem definiert wird, welches Bit als Ausgang, und welches als Eingang betrieben werden soll. Da der Befehl nach dem ersten Steuerwort bereits erwartet wird, ist hierfür keine Kennung des Befehls erforderlich.

Die Ausgabe setzt sich hierfür wie folgt zusammen:



PIO	Port	Binärkode	Hexcode
1	A	00000000	00
1	B	00000000	00
2	A	11011011	DB
2	B	01001001	49
3-7		11111111	FF
7	B	00000001	01

Nach diesem Schema ergeben sich durch unsere Hardware nebenstehende Ein- und Ausgabedefinitionen der einzelnen PIO-Ports.

Nachdem nun alle PIO's konfiguriert wurden, müssen auch noch die Displays initialisiert werden. Die nachfolgend beschriebenen Steuerwörter werden mit Hilfe des Moduls ,lcd()' übermittelt.

FUNCTION SET	→ 30 _h	(8-Bit Daten, 1-zeiliges Display, 5x7 Font)
DISPLAY ON/ OFF	→ 0C _h	(Display ,ein', Cursor ,aus', Cursor-Blinken ,aus')
CLEAR DISPLAY	→ 01 _h	(Display löschen)
ENTRY MODE SET	→ 06 _h	(Cursor auto-increment)

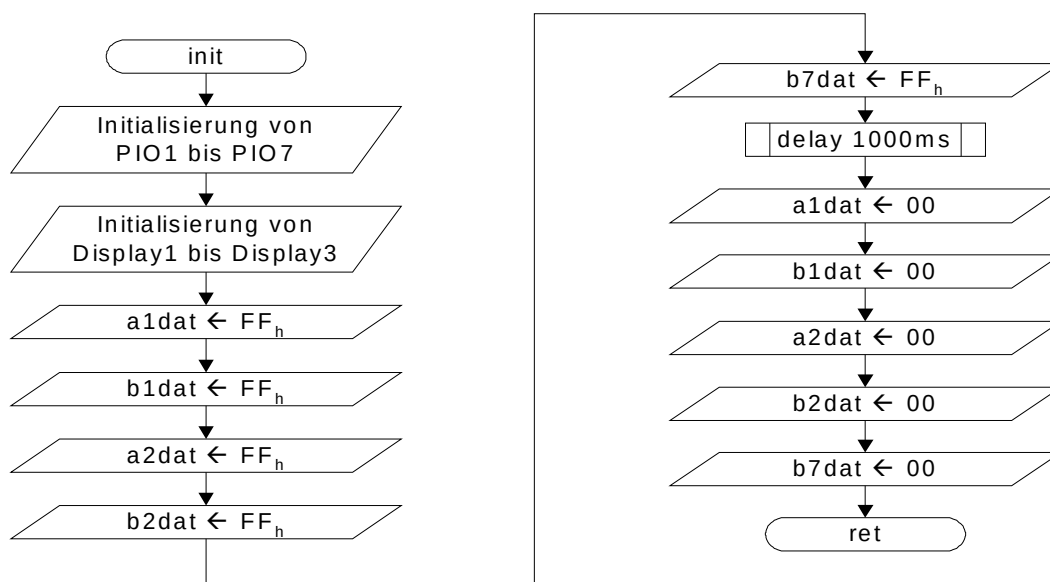
Sind diese Befehle an die 3 Displays übertragen, ist die eigentliche Konfiguration der Peripherie abgeschlossen.

Im Anschluss daran werden alle Ausgänge der PIO's auf High gesetzt. Dadurch erleuchten die LED's, als auch die LCD-Beleuchtung und der akustische Signalgeber ertönt.

Nach einer Zeit von einer Sekunde werden, verursacht durch den Aufruf von ,delay()', alle zuvor eingeschalteten Bauteile wieder ausgeschaltet.

Die Initialisierungsschleife ist somit beendet.

5.7.1.2 Programmablaufplan ,init()'



5.7.2 iniim2()

5.7.2.1 Programmbeschreibung

In dieser Funktion wird die PIO 2 für die Interruptverarbeitung initiiert. Dies ist ein unerlässlicher Schritt, der vor dem Anwenderprogramm erfolgen muss, da das Betätigen sämtlicher Taster auf der Frontplatte des Geräts sonst keine Aktionen auslösen würde. Zunächst wird die Interruptverarbeitung generell abgeschaltet. Dies erfolgt mit dem Befehl „di“ (disable interrupt). Nun muss im ersten Schritt das Interruptsteuerwort an die Control-Adresse von Port A gesendet werden. Dieses Steuerwort setzt sich nach unseren Bedürfnissen wie folgt zusammen: Nach der Übertragung des Interruptsteuerworts, wie bereits bei der Erläuterung des vorherigen Steuerworts angegeben, muss unmittelbar die Interruptmaske folgen. Hiermit wird der PIO mitgeteilt, welche Bits des angesprochenen Ports in der Lage sein sollen einen Interrupt auszulösen. Die Definition dieses Worts geschieht, ähnlich der Definition des Bitsteuerworts bei der PIO-Betriebsart ‚Bitcontrol‘. Trägt das entsprechende Bit eine 0, so signalisiert dies, ein Interruptaufruf ist möglich. Man spricht auch vom „Monitoren“ (überwachen) des Bits. Eine 1 bedeutet wiederum, dass über das Bit kein Aufruf erfolgen kann.

Somit ergeben sich folgende Interruptmasken für unser System:

PIO 2, Port A → 24_h

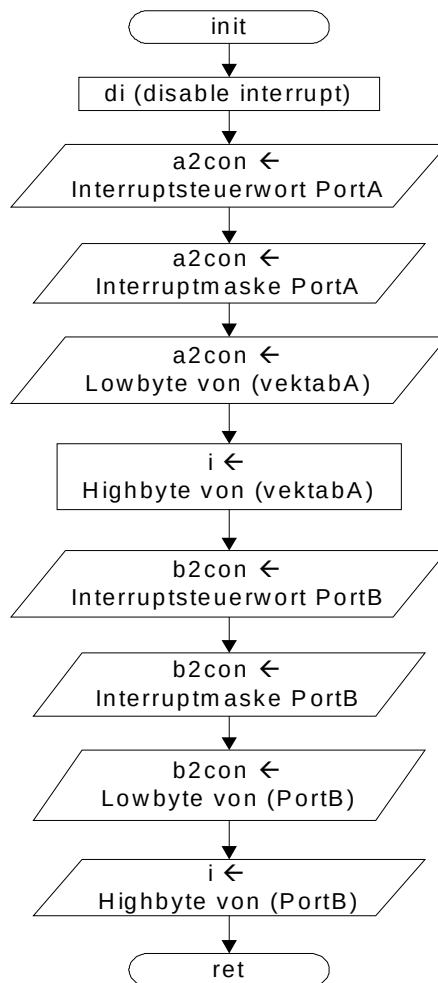
PIO 2, Port B → B6_h

Ist die Maskierung erfolgt, muss der PIO 2 das Low-Byte der Vektoradresse zugeteilt werden. Die Vektoradresse ist diejenige, zu der im Falle eines Interrupts gesprungen wird. Es ist eine Routine, die häufig „Interrupt-Routine“ betitelt wird. In unserem Fall heißt sie ‚irouta()‘ für Port A. Für Port B wurde sie ‚iroutb()‘ genannt. Die Vorbereitung des PIO-Ports ist damit abgeschlossen. Allerdings muss für eine einwandfreie Verarbeitung eines Interrupts auch noch das High-Byte der zuvor erläuterten Vektoradresse im Indexregister ‚I‘ abgespeichert werden. Die gleiche Prozedur erfolgt anschließend mit den oben erläuterten Steuerwörtern für Port B der PIO 2. Die Zeile *„intseg segment privat code para“* bedeutet folgendes:

Das erste Wort, intseg, dient lediglich als Layer (Name) für das folgende Codesegment, welches ausschließlich vom eigenen Programm verwendet werden kann. Das Segment ist vom Charaktertyp ‚para‘, das bedeutet es wird im Speicher ein zusammenhängender Bereich mit 16 Speicherplätzen eingeräumt. Dies ist notwendig, da die Interruptroutinen mit demselben Speicherplatzanfang beginnen müssen.

Die Initialisierung der interruptauslösenden PIO ist hiermit abgeschlossen.

5.7.2.2 Programmablaufplan ,iniim2()'



5.7.3 typ1()

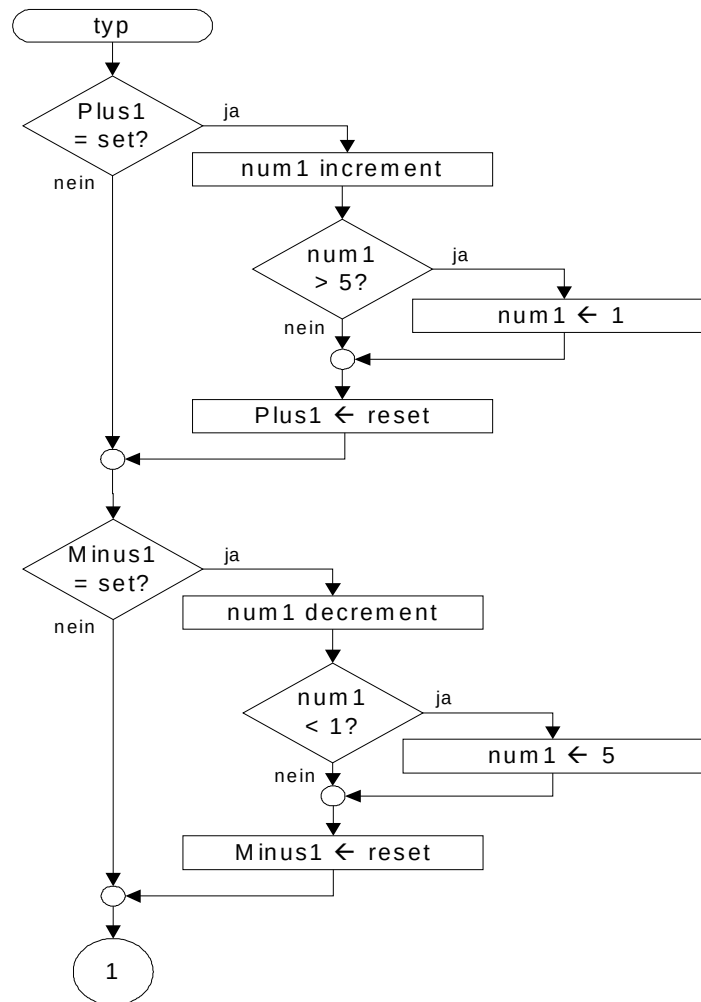
5.7.3.1 Programmbeschreibung

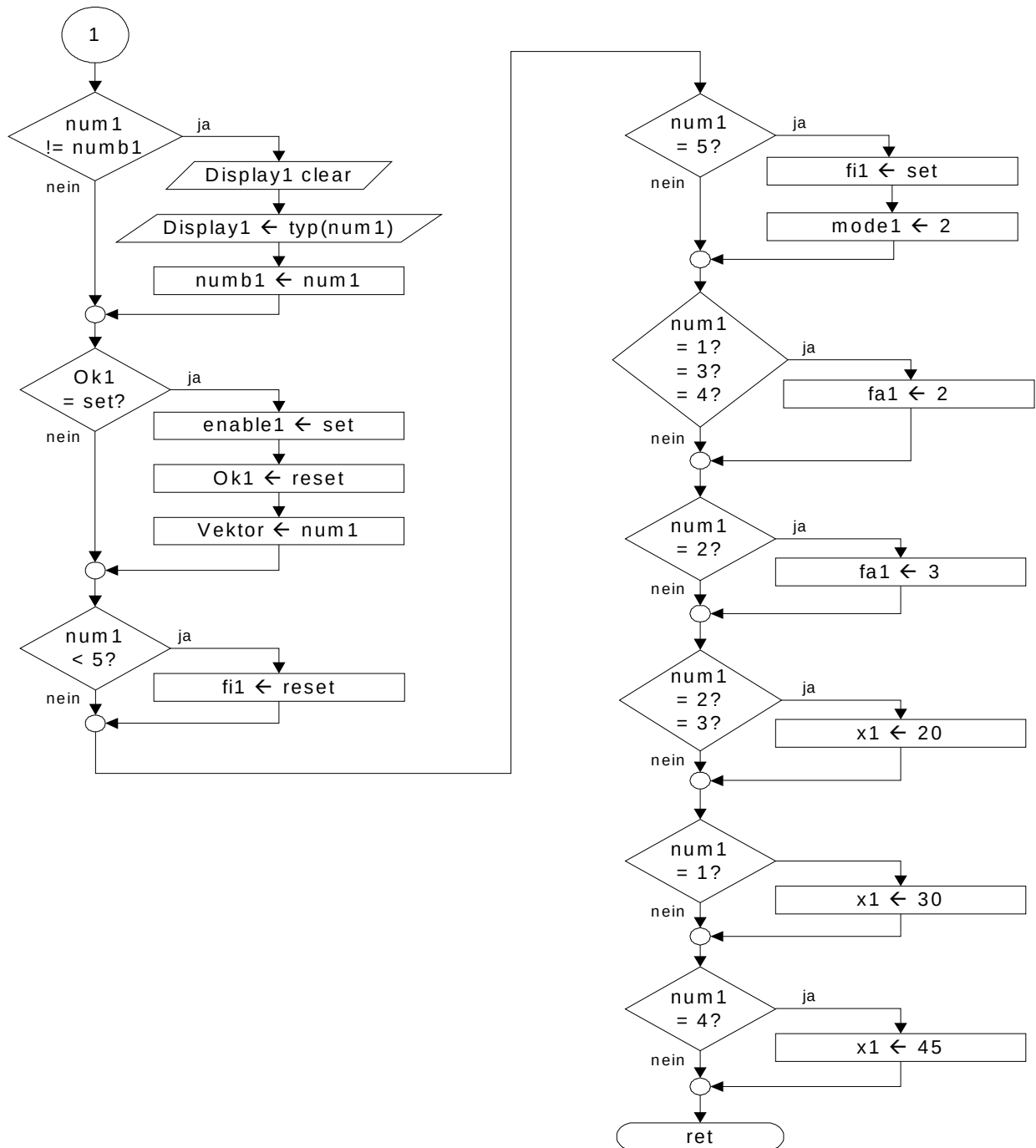
In dieser Funktion kann gewählt werden, welche Gerätevariante verwendet wird. Die Wahl ist bindend für den gesamten Trupp. Zu Beginn wird überprüft, ob die Taste Plus1 betätigt wurde. Ist dies der Fall, so wird der Funktionsinterne Zeiger ,num1' inkrementiert. Ist dieser nun größer als 5, wird er zurück auf den Wert 1 gesetzt. Ist dies geschehen, oder wurde die Taste nicht betätigt, erfolgt die gleiche Abfrage mit ,Minus1'. In diesem Fall wird ,num1' jedoch dekrementiert. Ist er unter 1, so wird er auf den Wert 5 gesetzt. Anschließend wird kontrolliert, ob der Zeiger ,num1' ungleich der Variable ,numb1' ist. Trifft dies zu, wird auf dem Display der entsprechende Gerätetyp, auf den der Zeiger ,num1' in der Vektorvariable ,typ[]' zeigt, ausgegeben. Die Variable ,numb1' wird nun mit dem Inhalt von ,num1' überschrieben. Sind ,num1' und ,numb1' bereits zuvor identisch, findet keine Ausgabe statt. Jetzt wird überprüft, ob ,Ok1' gesetzt ist. Ist dies der Fall, wird die Schleifenfreigabe gesetzt, die Variable ,Ok1' zurückgesetzt und der Inhalt von ,num1' (ausgewählter Gerätetyp) in die Speichervariable ,Daten[]' geschrieben.

Hat ,num1' einen Wert <5, so wird ,fi1' zurückgesetzt. Ist der Wert gleich 5, entspricht dies der Wahl der Variante ,Filter'. Somit wird der Merker ,fi1' gesetzt und ,mode1' auf 2 gestellt. Im Folgenden wird die Anzahl der Truppmitglieder definiert. Ist ,num1' gleich 1 | 3 | 5 wird ,fa1' mit 2 beschrieben, es gehören 2 FA zum Trupp. Hat ,num1' den Wert 2, gehören 3 FA dem Trupp an.

Als letzte Aktion in diesem Modul muss nun noch die maximale Einsatzzeit bestimmt werden, die der Trupp im Einsatz sein darf. Auch dies geschieht über eine Abfrage der Variable ,num1'. Beim Inhalt 2 | 3 wird der Grenzwert von 20 in ,x1' gespeichert. Ist der Wert 1, so wird ,x1' gleich 30 und bei der 4 erhält sie den Wert 45.

5.7.3.2 Programmablaufplan ,typ1()'





5.7.4 pid()

5.7.4.1 Programmbeschreibung

Diese Funktion wird in jedem Fall aufgerufen, nachdem eine Gerätevariante ausgewählt wurde. Zu Beginn des Programms werden die 3 Variablen ‚T1M1‘, ‚T1M2‘ sowie ‚T1M3‘ mit FF_h überschrieben. Anschließend wird der PID-Stick für das erste Truppmitglied eingelesen. Direkt im Anschluss wird dieses Datenwort mit 80_h maskiert. Dies muss erfolgen, da von den PID-IN-Platinen nur die ersten sieben Datenleitungen für die Datenübertragung verwendet werden. Hat die Variable nun einen Wert ungleich FF_h, wird die ‚led11(‘)‘ aufgerufen. Diese Vorgehensweise wird im Anschluss für die übrigen beiden Truppmitglieder wiederholt.

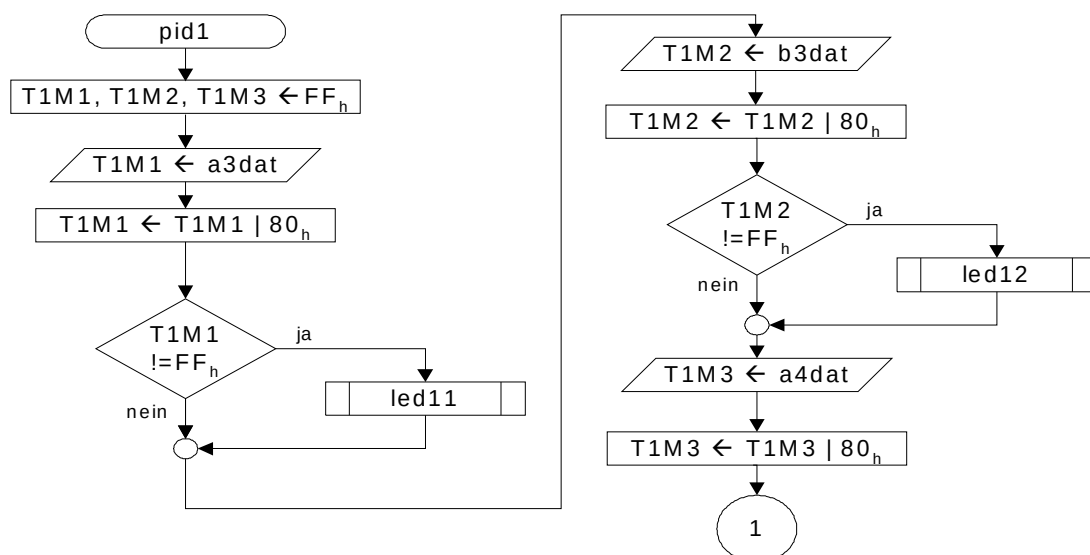
Im Anschluss folgt die Abfrage, wie viele Truppmitglieder dem Trupp angehören. Sind es drei, ‚fa1‘ hat den Wert 3, und sind ‚T1M1‘, ‚T1M2‘ sowie ‚T1M3‘ ungleich FF_h, so erfolgt die Schleifenfreigabe. Die eingelesenen PID-Stick-Daten werden nun in den Vektorspeicher geschrieben.

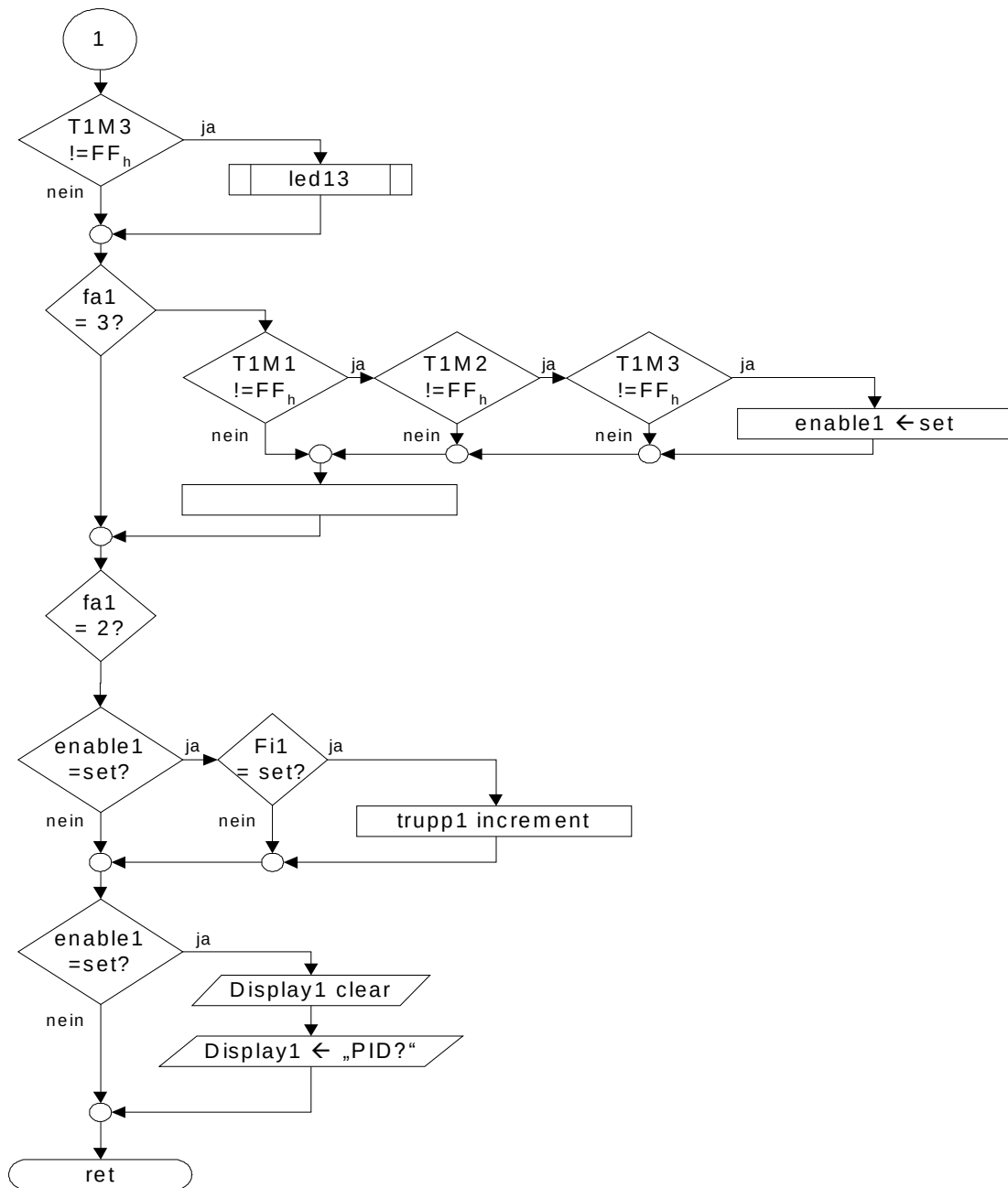
Trägt ‚fa1‘ den Wert 2, so erfolgt eine identische Kontrolle der Variablen und Abspeicherung der Daten. In diesem Fall werden jedoch nur die ersten beiden Variablen berücksichtigt.

Ist die Schleifenfreigabe gegeben und wurde die Gerätevariante „Filter“, ‚fi1‘ ist gesetzt, gewählt, wird nun automatisch der Schleifenzeiger manipuliert, da im Anschluss an diese Funktion keine Eingabe von Behälterdrücken erfolgen wird.

Als letzte Aktion wird das Display gelöscht und mit der Ausgabe „PID?“ neu beschrieben, sofern keine Schleifenfreigabe erfolgt.

5.7.4.2 Programmablaufplan ‚pid()‘



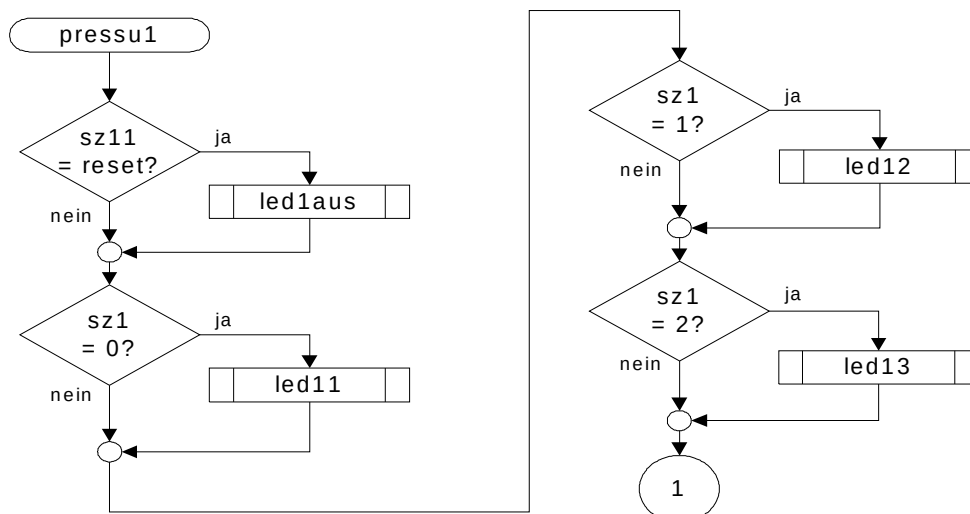


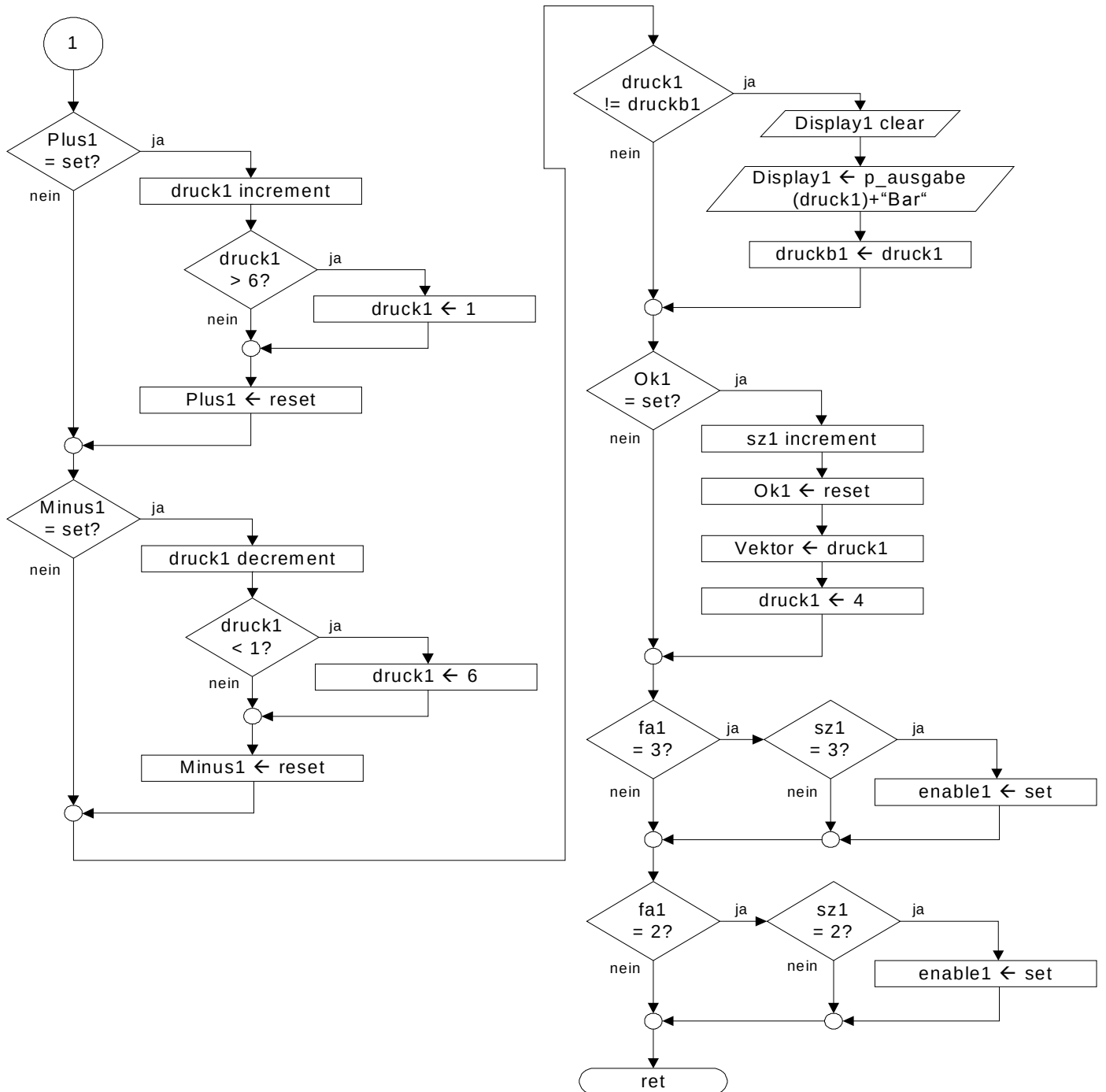
5.7.5 pressu()

5.7.5.1 Programmbeschreibung

Außer bei der Gerätevariante „Filter“ wird dieses Modul nach dem Einlesen der PID-Sticks benötigt. Die Behälterdrücke der FA werden aufgenommen. Im ersten Aufruf dieser Funktion wird das Unterprogramm ‚led1aus()‘ aufgerufen, da ‚sz11‘ zurückgesetzt ist. Anschließend wird der interne Schleifenzähler ‚sz1‘ überprüft. Trägt er die 0, wird ‚led11()‘ aufgerufen, bei der 1 die Funktion led12() und beim Wert 2 das Modul ‚led12()‘. Dadurch wird gewährleistet, dass vom Bediener permanent eingelesen werden kann, wessen Druck gerade abgefragt wird. Nun erfolgt die Frage, ob ‚Plus1‘ bereits gesetzt wurde. Ist dies der Fall, so wird die Variable ‚druck1‘ inkrementiert. Trägt diese nun einen Wert größer 6, so wird ‚druck1‘ mit 1 beschrieben. Der Merker ‚Plus1‘ wird gelöscht und die Abfrage ist beendet. Sollte nun ‚druck1‘ einen anderen Inhalt als ‚druck1‘ aufweisen, so wird das Display1 gelöscht und mit dem aktuellen Wert des Zeigers ‚druck1‘ auf der Variable ‚p_ausgabe[]‘ ausgegeben. Die Einheit „Bar“ wird unmittelbar angehängen und ‚druckb1‘ wird ‚druck1‘ angeglichen. Wurde bis zum jetzigen Zeitpunkt ‚OK1‘ gesetzt, wird zunächst ‚sz1‘ inkrementiert (Abfrage dieser Variable zu Beginn des Moduls) und ‚OK1‘ unmittelbar zurückgesetzt. Der momentane Zustand von ‚druck1‘ wird in den Vektorspeicher geschrieben und anschließend mit dem Startwert 4 geladen. Abschließend wird kontrolliert, ob der interne Schleifenzähler ‚sz1‘ dem Wert von ‚fa1‘ (Anzahl der Truppmitglieder) entspricht. Ist dem so, wird die Schleifenfreigabe gesetzt und das Modul wird beendet.

5.7.5.2 Programmablaufplan ‚pressu()‘



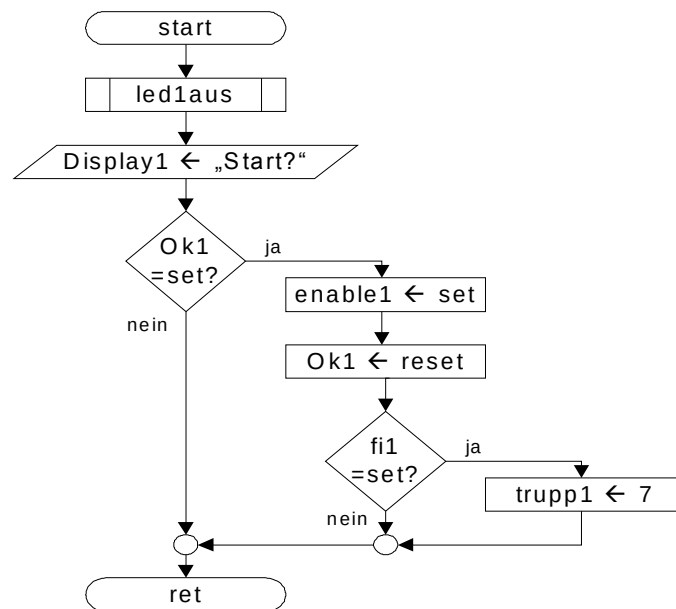


5.7.6 start()

5.7.6.1 Programmbeschreibung

Die Funktion ‚start()‘ dient der Signalisierung, dass die Vorbereitung für einen nachfolgenden Einsatz beendet ist. Um dies zu realisieren, wird auf dem Display „Start“ ausgegeben. Dies geschieht solange, bis die Taste ‚OK1‘ gedrückt wird. Anschließend wird mittels if-Abfrage kontrolliert, ob die Gerätevariante Filter gewählt wurde. Ist dies der Fall, so wird der Schleifenzeiger des jeweiligen Trupps manipuliert, um anschließend zum eigentlichen Zeitzähler für diese Gerätevariante zu gelangen. Ist dies nicht der Fall, wird die Funktion beendet und beim nächsten Schleifendurchgang erfolgt der Aufruf der Funktion ‚count11‘.

5.7.6.2 Programmablaufplan ‚start()‘



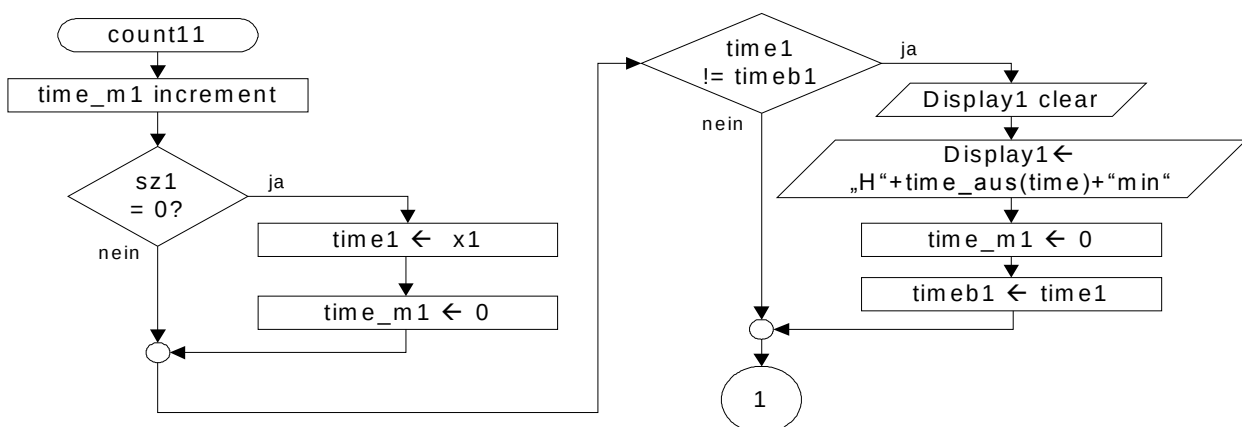
5.7.7 count11()

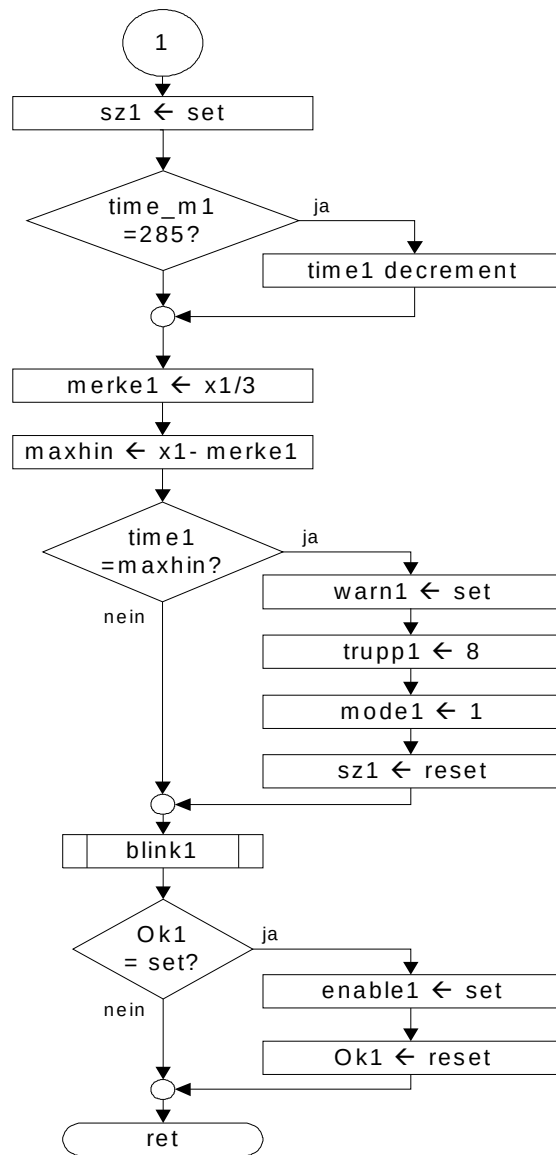
5.7.7.1 Programmbeschreibung

Dieses Unterprogramm ist der Einsatzzeitcounter für den Hinweg zur Einsatzstelle. Ist die intern verwendete Variable ‚sz1‘ noch auf 0, so wird die Funktion im aktuellen Einsatz zum ersten Mal aufgerufen. Somit muss die maximale Einsatzzeit von ‚x1‘ nach ‚time1‘ geschrieben werden. Der Merker ‚time_m1‘ wird zurückgesetzt. Hat ‚time1‘ nun einen anderen Inhalt als ‚timeb1‘, erfolgt die Ausgabe der aktuellen Zeit. Dies geschieht durch Löschen des Displays und anschließender Ausgabe von „H“, sowie dem Zeitwert (Variable ‚time_aus‘) und dem Anhang „min“. Danach wird ‚time_m1‘ erneut auf 0 gesetzt und ‚timeb1‘ an ‚time1‘ angepasst. Anschließend wird der Schleifenzähler ‚sz1‘ gesetzt. Dies bedeutet, die Funktion ist bereits einmal aufgerufen worden.

Hat ‚time_m1‘ den Inhalt 285, wird die Einsatzzeit in ‚time1‘ um 1 verringert. Im Anschluss daran wird der maximale Hinweg berechnet und in die Variable ‚maxhin1‘ übergeben. Es folgt die Abfrage, ob ‚time1‘ diesem Wert entspricht. Ist das der Fall, wird der Warnungsmerker ‚warn1‘ gesetzt. Des Weiteren wird der globale Schleifenzeiger manipuliert und auf 8 gestellt. Die Variable ‚mode1‘ bekommt den Inhalt 1 und der interne Zähler sz1 wird auf 0 gesetzt. Unabhängig von den vorherigen Abfragen wird das Modul ‚blink1()‘ aufgerufen. Wurde ‚OK1‘ betätigt, wird die Schleifenfreigabe erneut gesetzt und ‚Ok1‘ im Anschluss daran unmittelbar zurückgesetzt.

5.7.7.2 Programmablaufplan ‚count11()‘





5.7.8 ziel()

5.7.8.1 Programmbeschreibung

Bei Meldung des vorgehenden Trupps, dass er den tatsächlichen Einsatzort erreicht hat, erfolgt zunächst die Eingabe der Behälterdrücke. Daraus resultierend wird in diesem Unterprogramm die am Einsatzort zu verbleibende Zeit ermittelt. Zu Beginn dieses Moduls erfolgt der Aufruf von ‚led1aus()‘, sowie das Löschen des Displays mit anschließender Ausgabe „Ziel an“, solange die Variable ‚sz12‘ zurückgesetzt bleibt.

Ist dies nicht mehr zutreffend, schließt sich eine Reihe von Vergleichen und resultierenden Abfragen an.

Zunächst erfolgt die Kontrolle der Variable ‚sz11‘. Je nach Wert wird die dementsprechende Unterfunktion (led11, led12, led13) angesprochen. Im Anschluss daran wird die Variable ‚Plus1‘ ausgewertet. Ist diese gesetzt, so wird ‚z1‘ inkrementiert, beim Überschreiten des Grenzwertes 30 wird die Variable automatisch mit 12 beschrieben. ‚Plus1‘ wird jetzt wieder zurückgesetzt. Es folgt die Kontrolle von ‚Minus1‘. Bei gesetztem Status wird ‚z1‘ dekrementiert. Unterschreitet ‚z1‘ dabei den Wert 12, folgt automatisch die Korrektur auf 30. ‚Minus1‘ erhält nun wieder den Wert 0.

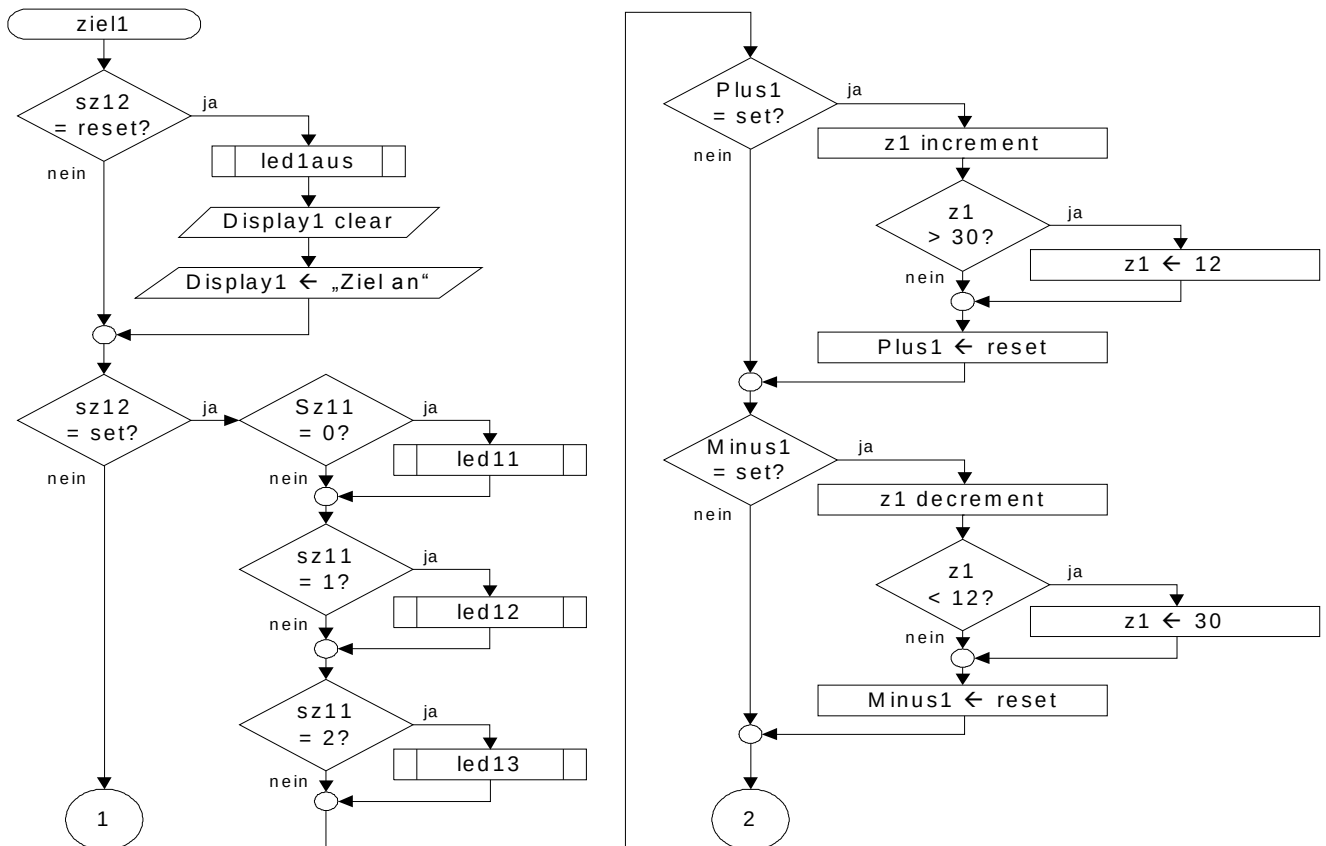
Bei Ungleichheit zwischen ‚zb1‘ und ‚z1‘ erfolgt nun die Ausgabe des Drucks (‚ziel_aus‘), auf welchen der Zeiger ‚z1‘ zeigt. Zuvor wird das Display gelöscht. Nach der Ausgabe wird die Einheit „Bar“ hinzugefügt und der Abgleich von ‚zb1‘ an ‚z1‘ wird durchgeführt. An dieser Stelle im Programm ist die Abhängigkeit von ‚sz1L‘ beendet. Es wird nun verglichen, ob ‚sz1‘ kleiner 5 ist. So lange bleibt ‚sz12‘ zurückgesetzt und ‚sz1‘ wird inkrementiert. Bei Überschreiten des Schwellwerts 5 wird ‚sz12‘ gesetzt. Bei gesetztem Zustand von ‚Ok1‘ wird ‚sz11‘ inkrementiert und ‚Ok1‘ unmittelbar auf 0 gesetzt. Die momentane Druckauswahl wird im Vektorspeicher eingetragen und ‚z1‘ auf den Startwert 25 sowie ‚zb1‘ auf 0 gesetzt.

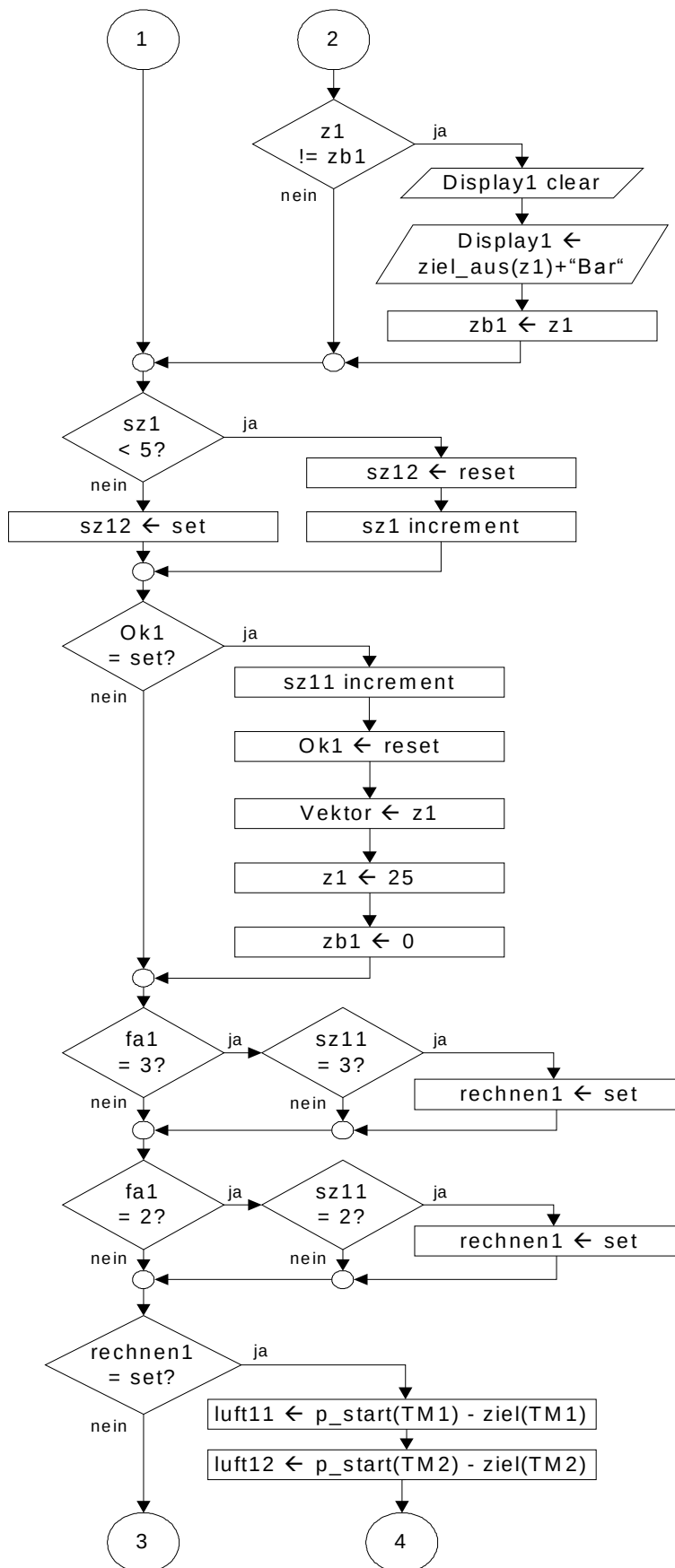
Zunächst werden jetzt ‚fa1‘ und ‚sz11‘ auf Gleichheit kontrolliert und bei Zutreffen wird ‚rechnen1‘ gesetzt. Dadurch wird der Teil der Funktion beendet, in welchem die Behälterdrücke eingegeben werden und es folgt die Ermittlung der Zeiten.

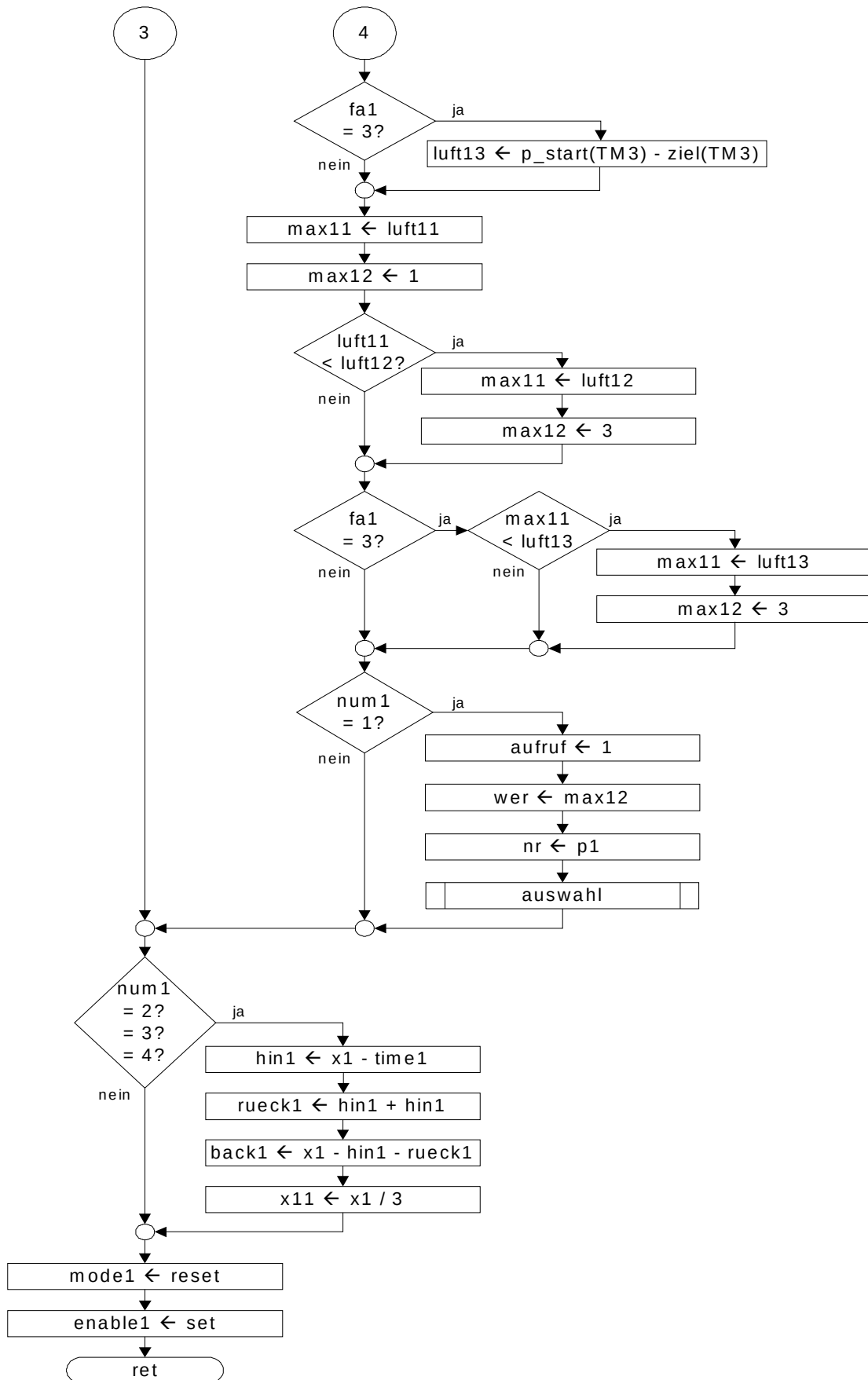
In Abhängigkeit vom gesetzten Zustand der Variable ‚rechnen1‘ wird nun zu Beginn der reale Luftverbrauch der Truppmitglieder berechnet und in den Variablen ‚luft11‘ und ‚luft12‘ eingetragen. Dies geschieht durch Subtraktion des Drucks bei Zielankunft (‚ziel[]‘) von Startdruck (‚p_start[]‘). Sind drei Mitglieder dem Trupp angehörig (‚fa1=3‘), findet diese Berechnung auch ein drittes Mal statt. Die Variable ‚max11‘, in der zum Ende dieser Funktion der größte Luftverbrauch steht, wird nun vorerst mit ‚luft11‘ beschrieben. In ‚max12‘ steht letztlich das Truppmitglied, welches den höchsten Luftverbrauch auf dem Hinweg hatte, erhält zunächst die logische 1.

Nun wird kontrolliert, ob ‚luft12‘ einen größeren Wert als ‚luft11‘ hat. In diesem Fall würden die soeben genannten Variablen mit den Werten des zweiten Truppmitgliedes überschrieben. Ist ‚fa1‘ gleich 3, so findet dieses Verfahren in gleicher Weise ein drittes Mal statt. Handelt es sich nun um einen Einsatz mit „PA 300“, so erhält ‚aufruf‘ die 1. Die Variable ‚wer‘ (verwendet in Modul ‚auswahl‘) bekommt den Zustand von ‚max12‘. Außerdem wird ‚ur‘ mit ‚p1‘ beschrieben und das Modul ‚auswahl()‘ wird verwendet. Unabhängig vom Zustand der Variable ‚rechnen1‘ wird nun noch ‚num1‘ auf Gleichheit mit 2, 3 und 4 überprüft. Tritt der Fall ein, finden folgende Berechnungen statt. Die Zeit für den Hinweg wird berechnet, indem die restliche Einsatzzeit ‚time1‘ von der gesamten Einsatzzeit ‚x1‘ subtrahiert und in ‚hin1‘ gespeichert wird. Die doppelte Zeit des Hinwegs wird als Maximum für den Rückweg in ‚rueck1‘ festgehalten. Die maximale Einsatzzeit an der Einsatzstelle wird in ‚back1‘ gespeichert und ergibt von ‚hin1‘ und ‚rueck1‘ von ‚x1‘. Die höchstzulässige Zeit für den Rückweg wird ebenfalls an dieser Stelle berechnet und in ‚x11‘ festgehalten, indem ‚x1‘ durch 3 dividiert wird. Abschließend wird ‚mode1‘ zurückgesetzt, es erfolgt die Schleifenfreigabe und das Modul ist abgeschlossen.

5.7.8.2 Programmablaufplan ‚ziel()‘





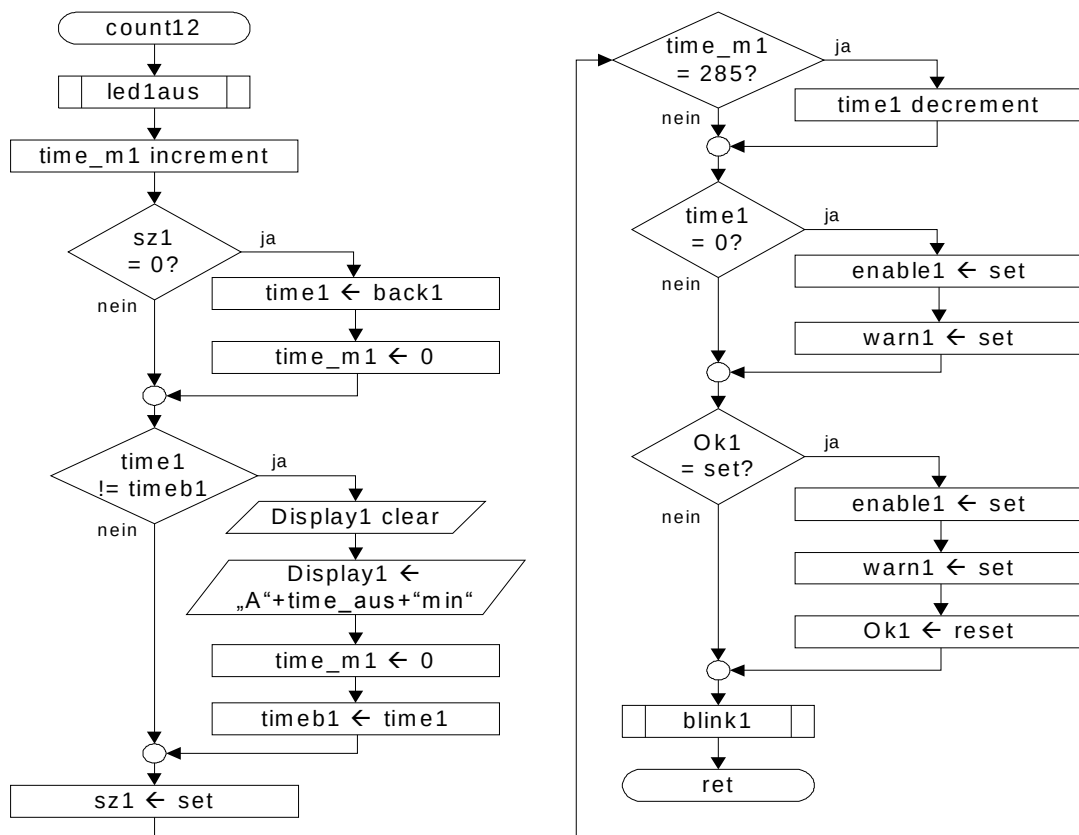


5.7.9 count12()

5.7.9.1 Programmbeschreibung

Der Aufruf dieses Unterprogramms erfolgt, nachdem die am Einsatzort zu verbleibende Zeit ermittelt wurde. Es wird die momentan restliche Einsatzzeit auf dem Display ausgegeben. Da der Aufbau dieser Funktion an den von ‚count11()‘ angelehnt ist, wird im Folgenden lediglich auf die Unterschiede eingegangen. Zu Beginn erfolgt der Aufruf ‚led1aus()‘. Dieses ist in ‚count11()‘ nicht erforderlich, da unmittelbar zuvor keinerlei LED’s angesteuert werden. Ein weiterer kleiner Unterschied ist, dass bei der Ausgabe der Zeit nicht ein „H“, sondern das „A“ vorangestellt ist. Im Wesentlichen sind 2 elementare Unterschiede der Funktionen festzustellen. Hat ‚sz1‘ den Wert 0, wird die runterzählende Zeit nicht mehr der gesamten Einsatzzeit, sondern der zuvor errechneten, an der Einsatzstelle zu verbleibenden Zeit angeglichen. Des Weiteren wird in dieser Funktion nicht kontrolliert, ob ‚time1‘ gleich ‚maxhin1‘ ist. Hier wird überprüft, ob der Inhalt 0 ist. Die Schleifenfreigabe sowie der Merker ‚warn1‘ werden automatisch gesetzt, wenn der Inhalt von ‚time1‘ gleich 0 ist.

5.7.9.2 Programmablaufplan ‚count12()‘



5.7.10 count13()

5.7.10.1 Programmbeschreibung

Diese Funktion ist dazu dienlich, die Zeit des „normalen Rückwegs“, sowie die des Rückwegs durch Abbruch zu bemessen und auszugeben. Wurde die Gerätevariante ‚Filter‘ selektiert, läuft die eigentliche Arbeitszeitbemessung hierüber ab. Ebenfalls wird bei Bedarf der zuvor gesetzte „Warnmerker“ ‚warn1‘ abgefangen und verarbeitet.

Erste Aktion in dieser Routine ist die Abfrage, ob die Variable ‚warn1‘ gesetzt wurde. Ist dies der Fall und ‚sz11‘ hat den Inhalt 0 sowie ‚mode1‘ ist gesetzt, wird das Display gelöscht und es folgt die Displayausgabe „Abbruch!“. Außerdem wird der global verwendete Merker ‚sz‘ mit 300 beschrieben. Ist ‚warn1‘ gesetzt und sowie ‚sz11‘ als auch ‚mode1‘ sind auf 0, wird dies ebenfalls gelöscht und die Ausgabe „Rueckweg“ folgt. Anschließend wird ‚timeb1‘ auf 0 gesetzt und der globale Merker ‚sz‘ mit 300 überschrieben. Nach beiden Varianten wird anschließend ‚sz11‘ gesetzt, sowie ‚bz‘ mit 2 beschrieben. Wurde in der Zwischenzeit bereits Ok1 betätigt, setzt das Programm ‚warn1‘ sowie ‚Ok1‘ unmittelbar zurück.

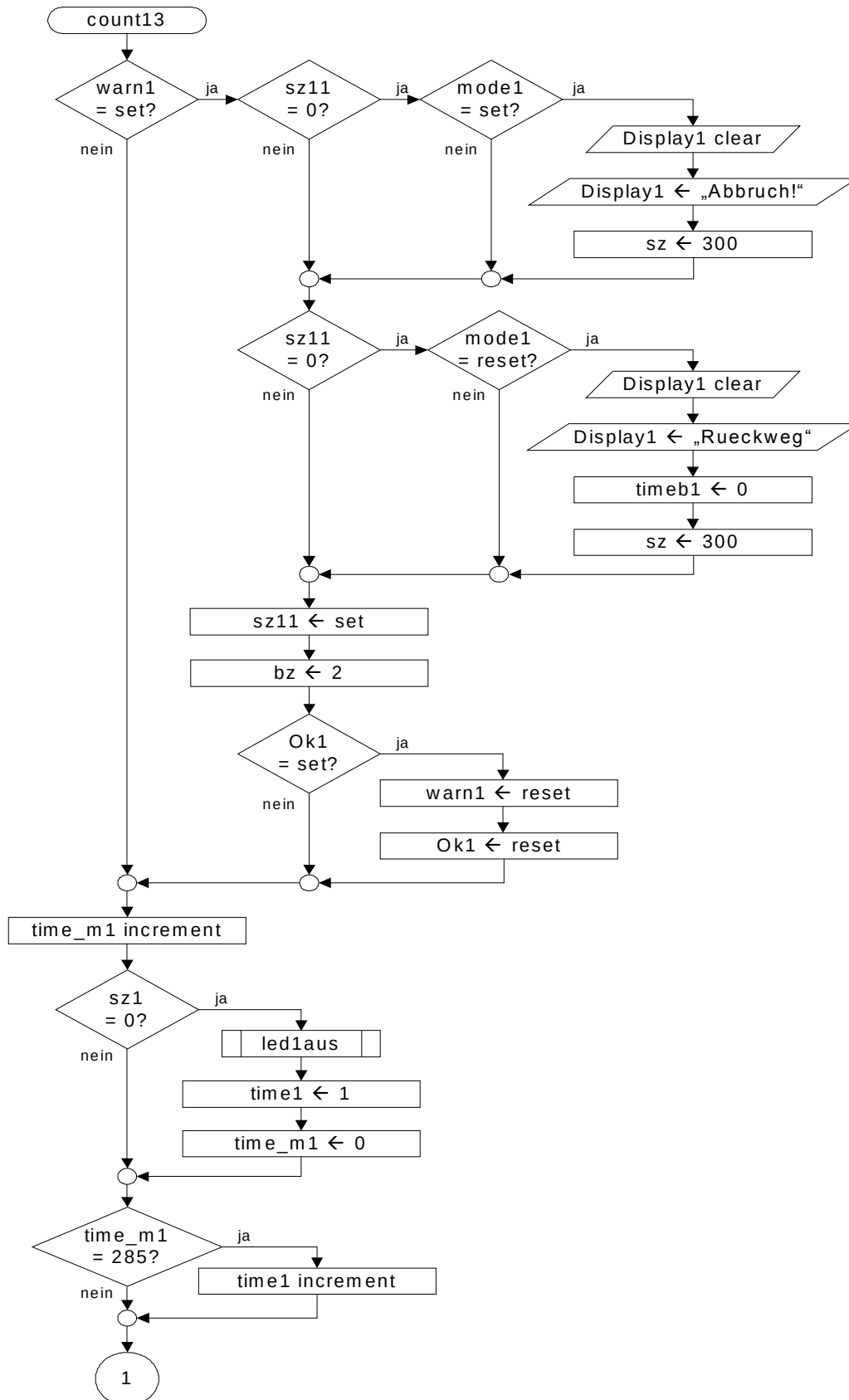
Damit ist die Verarbeitung von ‚warn1‘ abgeschlossen. Folgend wird, wie in der Funktion ‚count11()‘ beschrieben, ‚time_m1‘ je Funktionsaufruf inkrementiert. Im ersten Durchgang erfolgt das Setzen von ‚time1‘, sowie das Rücksetzen von ‚time_m1‘. Hat ‚time_m1‘ den Wert 285 erreicht, wird ‚time1‘ automatisch inkrementiert. Ist ‚warn1‘ sowie ‚notfall1‘ bereits zurückgesetzt oder die Variable ‚fi1‘ gesetzt, folgen die nächsten Abfragen. Bei Ungleichheit von ‚time1‘ und ‚timeb1‘, wird zunächst entschieden welches Vorzeichen für die Zeit auf dem Display ausgegeben wird. Die Anzeige von „Fi“ erfolgt, wenn ‚fi1‘ gesetzt wurde. Ist dies jedoch nicht der Fall, hängt das Vorzeichen von ‚mode1‘ ab. Bei dessen Inhalt 0 wird „R“, ansonsten „RA“ ausgegeben. Unabhängig vom angezeigten Vorzeichen folgt die Ausgabe der bisher benötigten Rückzugszeit. Anschließend wird, wie gewohnt ‚time_m1‘ auf 0 gesetzt, sowie ‚timeb1‘ an ‚time1‘ angeglichen.

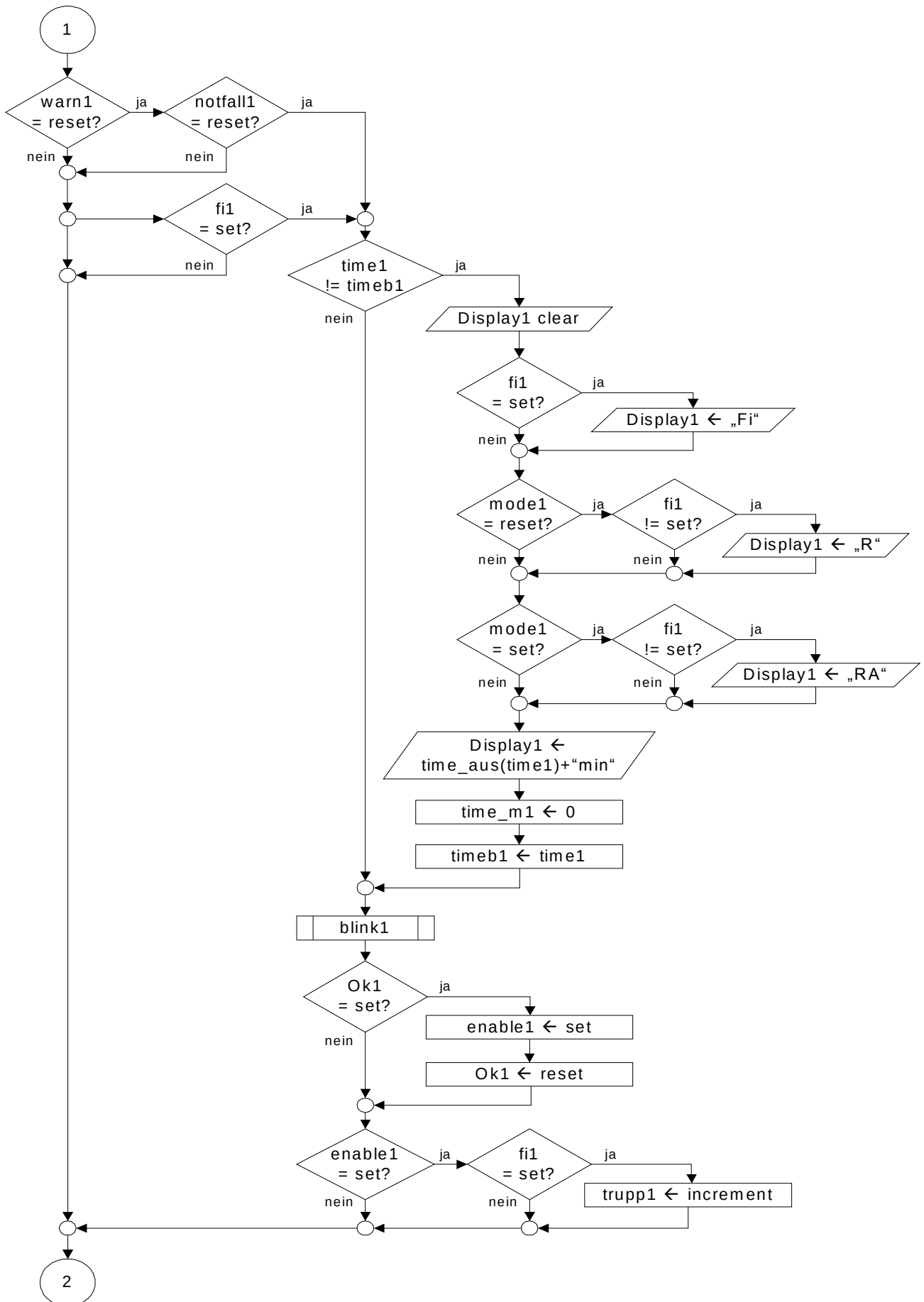
Nach diesen Ausgaben erfolgt zunächst der Aufruf von ‚blink1()‘. Danach wird ‚Ok1‘ abgefragt. Ist die Variable gesetzt, wird die Schleifenfreigabe erteilt und ‚Ok1‘ unmittelbar zurückgesetzt. Ist die Schleifenfreigabe sowie ‚fi1‘ gesetzt, wird erneut der Schleifenzeiger ‚trupp1‘ manipuliert, um die nächste Funktion zu überspringen. Unabhängig von ‚warn1‘ folgt nun die Kontrolle, ob ‚sz12‘ sowie ‚Ok1‘ gesetzt sind. Trifft dies zu, wird erneut die Schleifenfreigabe gesetzt und ‚Ok1‘ zurückgesetzt. Sind ‚notfall1‘ und ‚Ok1‘ gesetzt, wird automatisch der Merker ‚bz‘ (akustisches Warnsignal), sowie ‚Ok1‘ zurückgesetzt. Außerdem wird der interne Schleifenmerker

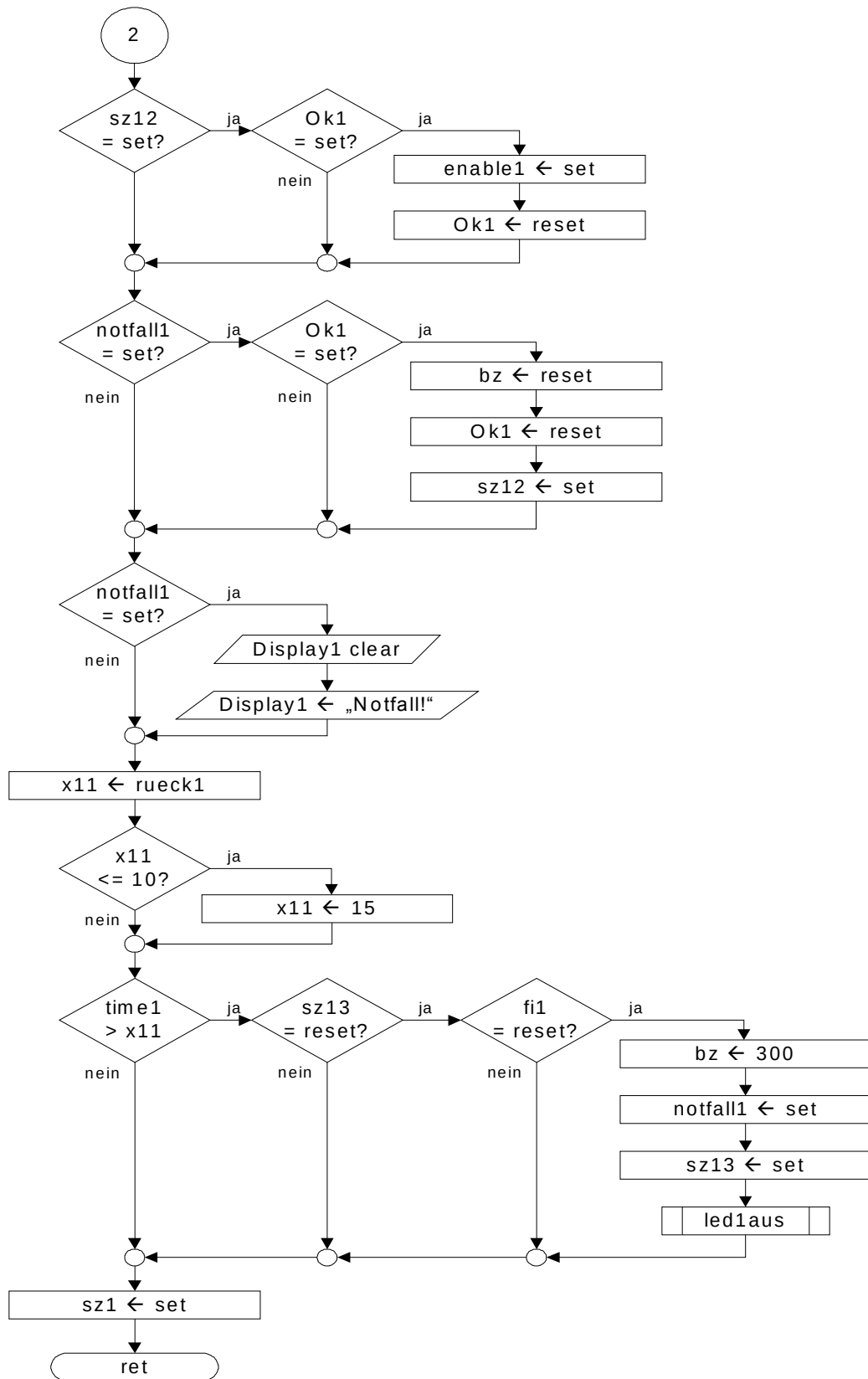
,sz12' gesetzt. Anschließend wird überprüft, ob ,notfall1' eine 1 beinhaltet, um in diesem Fall unmittelbar das Display zu löschen und die Displayausgabe „Notfall!“ zu veranlassen.

Danach wird im Programm die maximale Rückzugszeit ermittelt. Hat der aktuelle Zeitzähler diesen Wert überschritten und sind außerdem ,sz13' sowie ,fi1' zurückgesetzt, wird ,bz' mit 300 und ,notfall1' sowie ,sz13' mit 0 beschrieben. Der Aufruf von ,led1aus()' erfolgt. Abschließend wird nun noch die Variable ,sz1' auf 1 gestellt.

5.7.10.2 Programmablaufplan ,count13()'





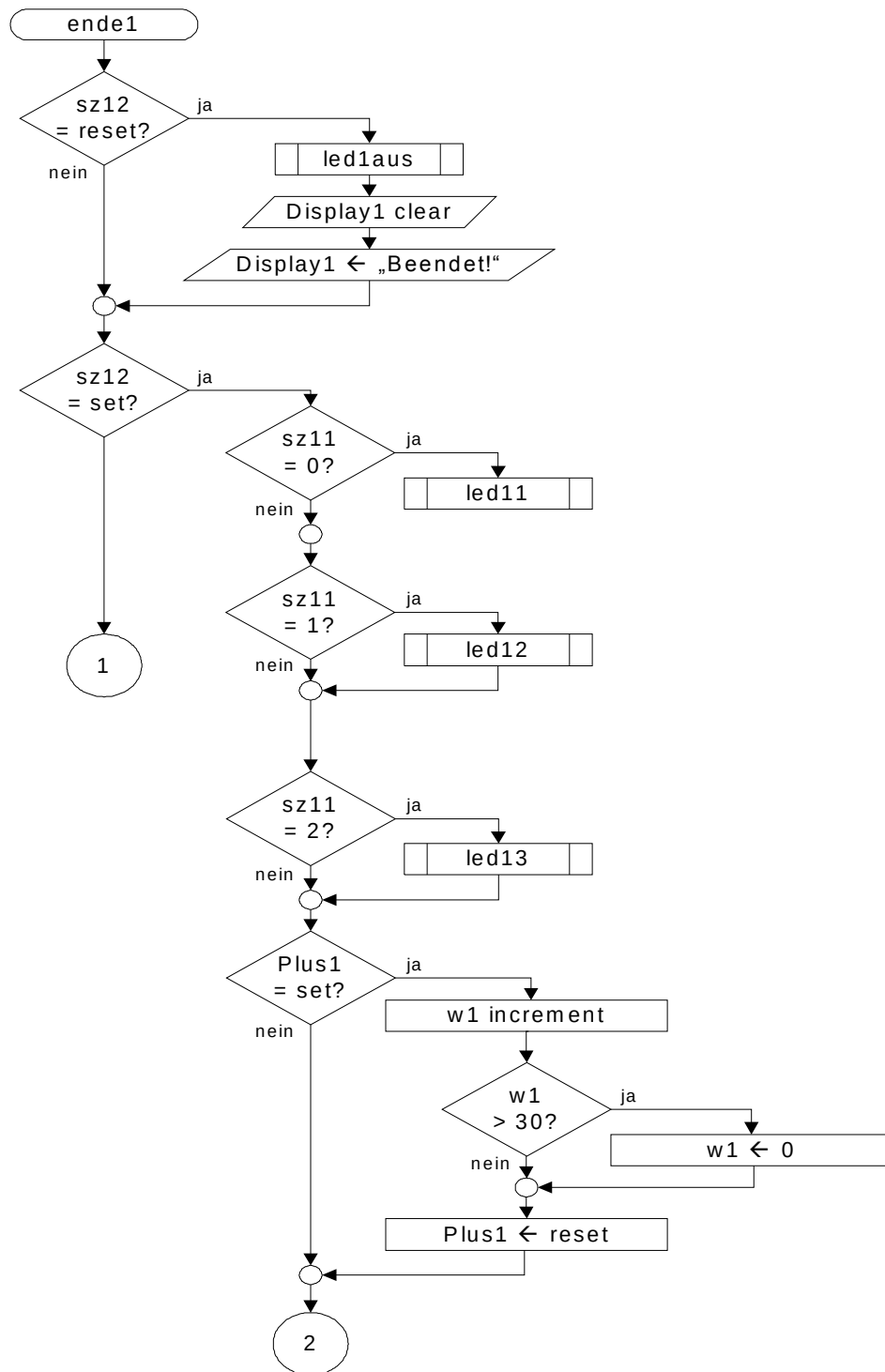


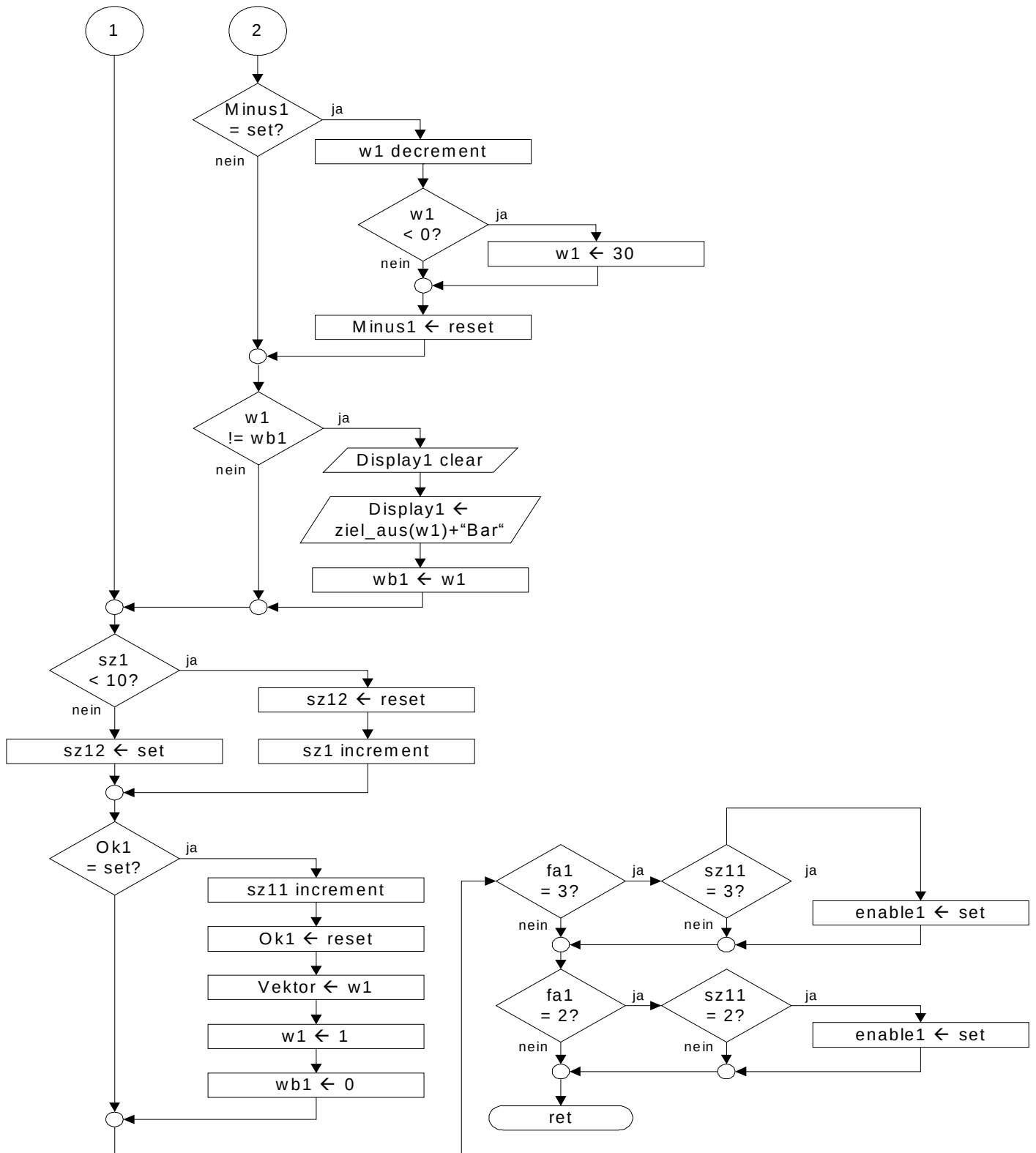
5.7.11 ende1()

5.7.11.1 Programmbeschreibung

Das Modul ‚ende1()‘ wird nach der Rückkehr des Trupps verwendet. Hierdurch wird die Eingabe des Behälterdrucks nach dem Einsatz realisiert. Zu Beginn wird solange ‚led1aus()‘ ausgeführt, sowie das Display gelöscht und „Beendet“ im Display ausgegeben, wie ‚sz12‘ zurückgesetzt ist. Ist dies nicht mehr der Fall, folgen diverse Aufrufe. In Abhängigkeit von ‚sz11‘ wird beim Wert 0 das Programm ‚led11()‘, bei 1 das Modul ‚led12()‘ und bei 2 das Modul ‚led13()‘ aufgerufen. Ist ‚Plus1‘ im Anschluss gesetzt, so wird ‚w1‘ inkrementiert. Wird dabei die Schwelle von 30 überschritten, erfolgt unmittelbar die Rückstellung auf 0. Plus1 wird automatisch zurückgesetzt. Im gesetzten Zustand von ‚Minus1‘ wird der Inhalt von ‚w1‘ um 1 verringert. Bei Unterschreitung der Grenze 0 wird der Wert automatisch wieder auf 30 gesetzt. Auch ‚Minus1‘ wird im Anschluss zurückgesetzt. Nach diesen Schritten wird nun, wenn ‚w1‘ ungleich ‚wb1‘ ist, das Display gelöscht und der gewählte Behälterdruckwert wird mit dem Anhang „Bar“ ausgegeben. Zusätzlich wird ‚wb1‘ mit dem Inhalt von ‚w1‘ überschrieben. Solange ‚sz1‘ kleiner als 10 ist, wird nun ‚sz12‘ zurückgesetzt und ‚sz1‘ inkrementiert. Ansonsten wird ‚sz12‘ gesetzt. Ist Taste OK1 betätigt worden, folgt die Inkrementierung von ‚sz11‘, ‚Ok1‘ erneut resetet und die gewählten Daten in die Speichervariable ‚Daten[]‘ übertragen. Außerdem erhält ‚w1‘ den Inhalt 1 und ‚wb1‘ die 0. Im Anschluss daran wird abschließend kontrolliert, ob ‚fa1‘ gleich ‚sz11‘ ist. Bei Zutreffen erfolgt die Schleifenfreigabe.

5.7.11.2 Programmablaufplan ,ende1()'



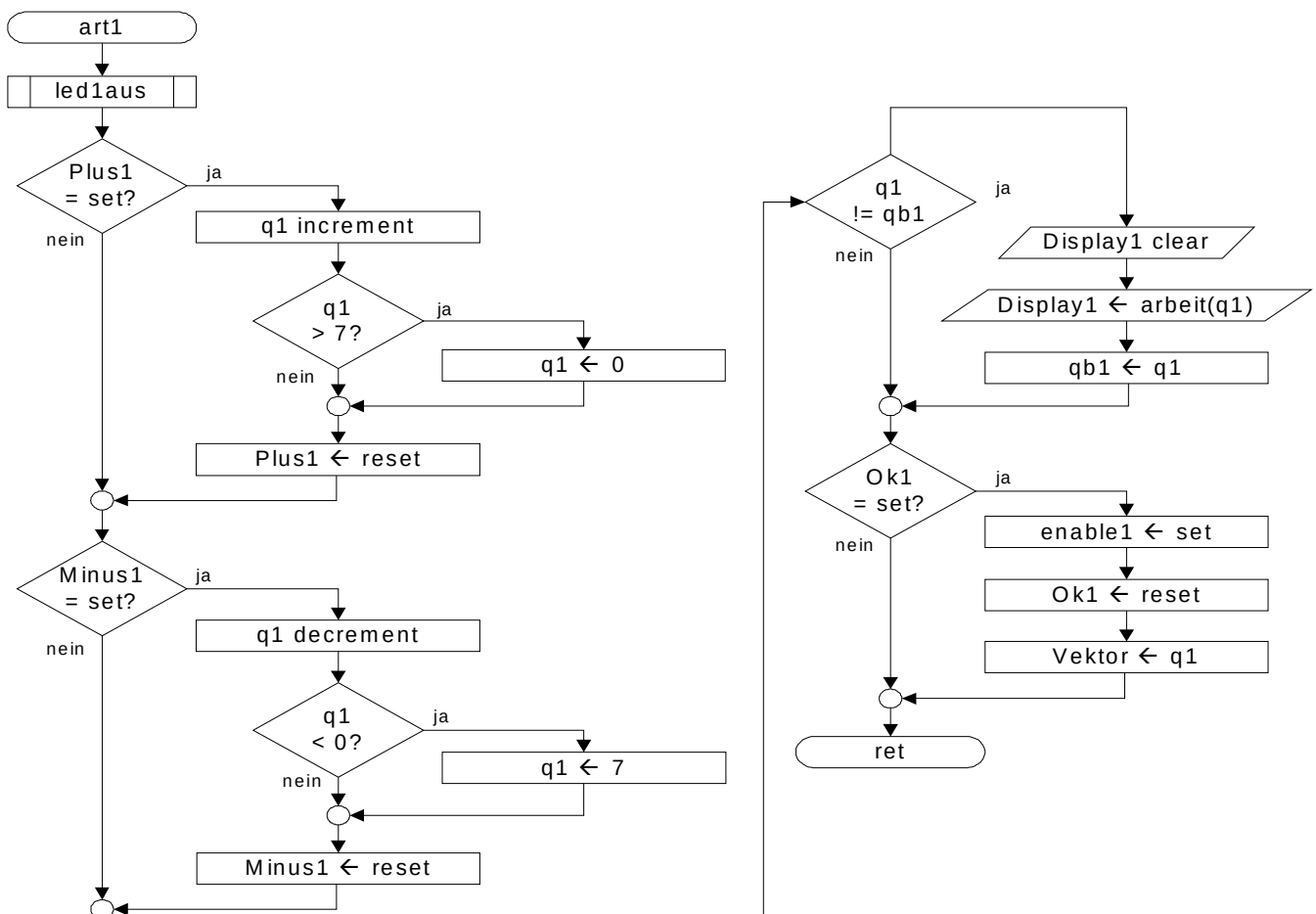


5.7.12 art1()

5.7.12.1 Programmbeschreibung

Dieses Modul dient der nach einem Einsatz erforderlichen Aufnahme der vom Trupp durchgeführten Tätigkeit. Zu Beginn des Moduls wird ‚led1aus()‘ aufgerufen, um alle LED's auszuschalten. Es wird kontrolliert, ob der Merker ‚Plus1‘ gesetzt ist. Trifft dies zu, wird die Variable ‚q1‘ inkrementiert. Hat sie nun einen Wert >7 wird sie automatisch auf 0 zurückgesetzt. Ist die Variable ‚Minus1‘ gesetzt, wird ‚q1‘ dekrementiert. Ist der Inhalt nun <0, wird er auf 7 gesetzt. ‚Minus1‘ wird nach dieser Überprüfung ebenfalls zurückgesetzt. Ist ‚q1‘ ungleich ‚qb1‘, wird das Display gelöscht und es wird die Tätigkeit ausgegeben, auf die ‚q1‘ gerade in der Vektorvariable ‚arbeit[]‘ zeigt. Ist dies geschehen, wird ‚qb1‘ an ‚q1‘ angeglichen. Wurde seit dem letzten Reset von ‚Ok1‘ erneut die Taste OK1 betätigt, wird die Schleifenfreigabe ‚enable1‘ gesetzt, ‚Ok1‘ wird zurückgesetzt und die gewählte Tätigkeit in die Speichervariable ‚Daten[]‘ für den gesamten Trupp übermittelt.

5.7.12.2 Programmablaufplan ‚art()‘



5.7.13 auswahl()

5.7.13.1 Programmbeschreibung

Dieses Unterprogramm ist für alle drei Truppmodule geeignet. Bei der Gerätevariante „PA 300“ wird es zur Ermittlung der restlichen Arbeitszeit nach Ankunft an der Einsatzstelle aufgerufen.

Zunächst wird überprüft, welcher „Startdruck“ relevant ist. Dazu wurde in der Funktion ‚ziel()‘ ermittelt, welches Truppmitglied den größten Luftverbrauch während des Hinwegs hatte. Dies ist das ausschlaggebende Truppmitglied.

Somit ist der relevante Startdruck ermittelt und die verbleibende Arbeitszeit kann definiert werden. Hierfür wird der Behälterdruck bei der Meldung „Ziel an“ abgefragt und anschließend mit Hilfe der folgenden Wertungstabellen, welche dem Programm zur Verfügung stehen, die Zeit ermittelt.

Startdruck von 300 Bar:

Ziel erreicht: (in Bar)	290	280	270	260	250	240	230	220	210	200
Rückmarsch nach: (in min)	29	28	25	22	18	14	11	7	4	sofort

Startdruck von 290 Bar:

Ziel erreicht: (in Bar)	280	270	260	250	240	230	220	210	200	190
Rückmarsch nach: (in min)	28	26	24	20	17	13	10	6	2	sofort

Startdruck von 280 Bar:

Ziel erreicht: (in Bar)	270	260	250	240	230	220	210	200	190	180
Rückmarsch nach: (in min)	26	25	23	19	16	12	8	5	1	sofort

Startdruck von 270 Bar:

Ziel erreicht: (in Bar)	260	250	240	230	220	210	200	190	180	170
Rückmarsch nach: (in min)	25	24	22	18	14	11	7	4	sofort	sofort

5.7.14 led11()

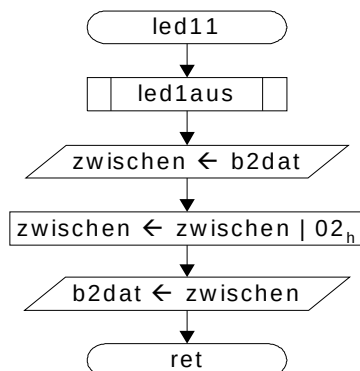
5.7.14.1 Programmbeschreibung

Dies Modul dient dazu, einzelne LED's einzuschalten und zuvor alle anderen des jeweiligen Trupps auszuschalten. Solche Module werden häufig auch „Treiber“ genannt. Es ergeben sich somit 9 LED-Treiber, je Trupp 3. Da die LED's einzeln leuchten sollen, werden zu Beginn alle anderen mit Hilfe des zugehörigen Moduls ausgeschaltet. Der entsprechende Port, an dem sich die Diode befindet, wird eingelesen, mit einer Maske ODER verknüpft und im Anschluss daran wieder zurückgegeben.

Folgende Masken werden für die Treiber verwendet:

Trupp	LED	Modul	Port	PIO	Maske
1	1	led11()	B	2	02
	2	led12()	A	2	04
	3	led13()	B	2	04
2	1	led21()	B	2	10
	2	led22()	A	2	20
	3	led23()	B	2	20
3	1	led31()	B	2	80
	2	led32()	B	1	01
	3	led33()	A	1	01

5.7.14.2 Programmablaufplan ,led11()'



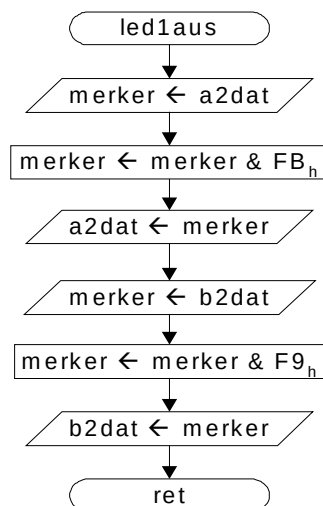
5.7.15 led1aus()

5.7.15.1 Programmbeschreibung

Ähnlich wie zum Einschalten der LED's, wird auch eine Funktion zum Ausschalten dieser benötigt. Hierfür werden ebenfalls die Port-Zustände in die Variable ‚merker‘ eingelesen, mit einer Maske UND verknüpft und wieder zum Port gesendet. Je nach Modul ergeben sich auch hier verschiedene Masken und einzulesende Ports:

Trupp	Modul	Port	PIO	Maske (Hex)	LED
1	led1aus()	A	2	FB	2
		B	2	F9	1, 3
2	led2aus()	A	2	DF	2
		B	2	CF	1, 3
3	led3aus()	B	2	7F	1
		A	1	FE	3
		B	1	FE	2

5.7.15.2 Programmablaufplan ‚led1aus()‘



5.7.16 blink1()

5.7.16.1 Programmbeschreibung

Hier handelt es sich um ein zentrales Modul, welches aus verschiedenen Unterprogrammen aufgerufen wird. Je nach Modul, aus welchem ‚blink()‘ aufgerufen wird, soll eine Blinkreihenfolge der drei Status-LED’s der Trupps erzeugt werden.

Durch jeden Aufruf des Programms wird zunächst der Merker ‚bm1‘ inkrementiert. Es folgt der Vergleich der Variable ‚trupp1‘. Hat sie den Inhalt 5, folgt eine Prozedur, die ein „Lauflicht nach rechts“ inszeniert. Realisiert wird dies durch den Vergleich der Variable ‚bm1‘ mit anschließendem Aufruf der LED-Steuermodule:

Wert: bm1 <5	Aktion:	led11()
Wert: bm1 >=5 & <10	Aktion:	led12()
Wert: bm1 >=10 & <15	Aktion:	led13()
Wert: bm1 >=15 & <20	Aktion:	led1aus()
Wert: bm1 >=20	Aktion:	bm1 ← reset

Hat ‚trupp1‘ den Wert 7, sollen die LED’s periodisch ein- und wieder ausgeschaltet werden. Hierfür wird zunächst kontrolliert, ob ‚bm1‘ kleiner als 10 ist. Trifft dies zu, werden alle LED’s durch ‚led1aus()‘ ausgeschaltet. Ist ‚bm1‘ größer als 10, wird der Zustand von ‚b2dat‘ eingelesen, anschließend mit 06h ODER verknüpft und wieder zum Port zurück gesendet. Dies geschieht danach ebenfalls mit ‚a2dat‘. Sobald ‚bm1‘ gleich 20 ist, wird die Variable zurück auf 0 gesetzt.

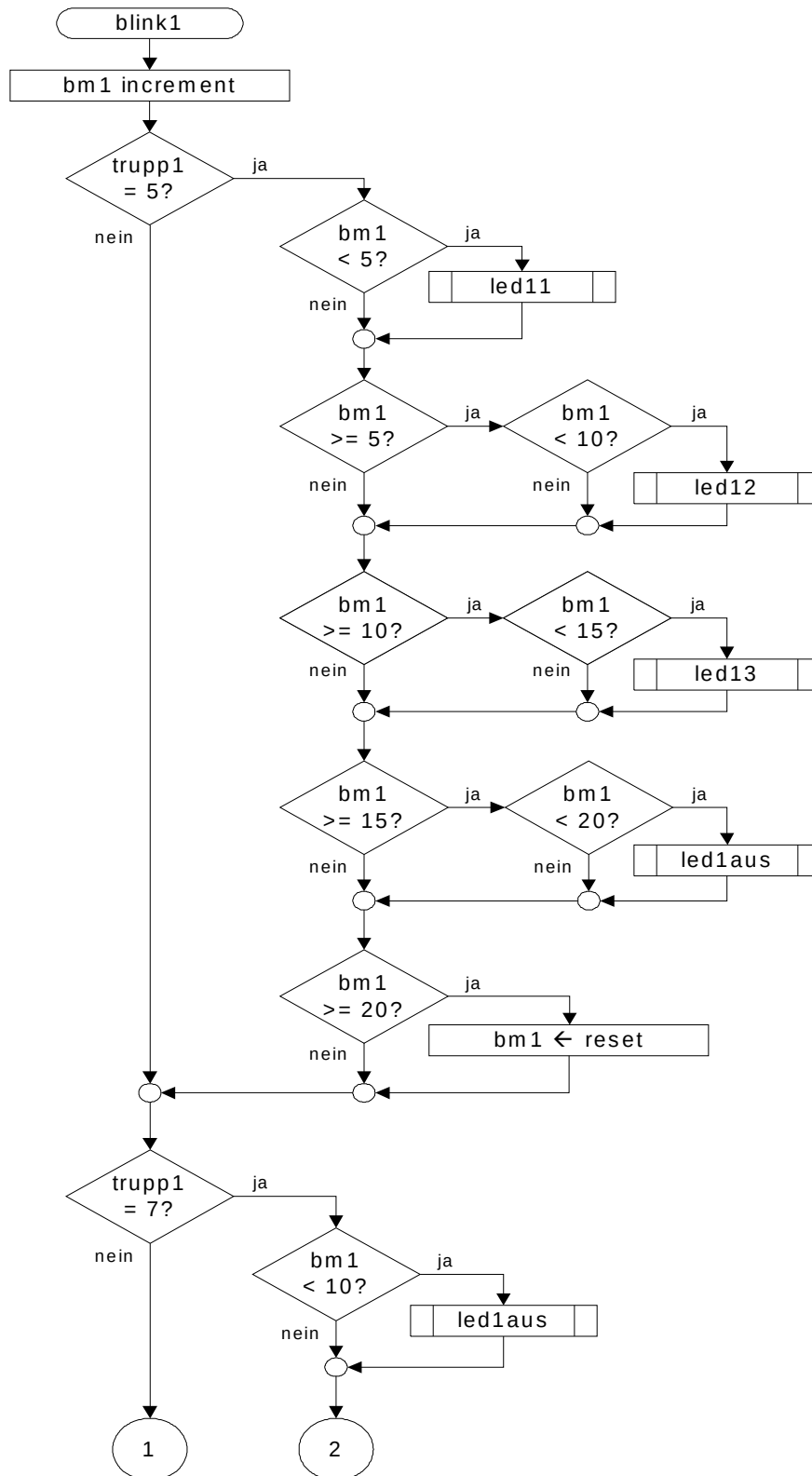
In den Modulen ‚blink2()‘ und ‚blink3()‘ werden die Ports mit anderen Werten maskiert, welche sich aus der Hardware ergeben. Bei ‚blink3()‘ sind hardwarebedingt auch andere Ports einzulesen.

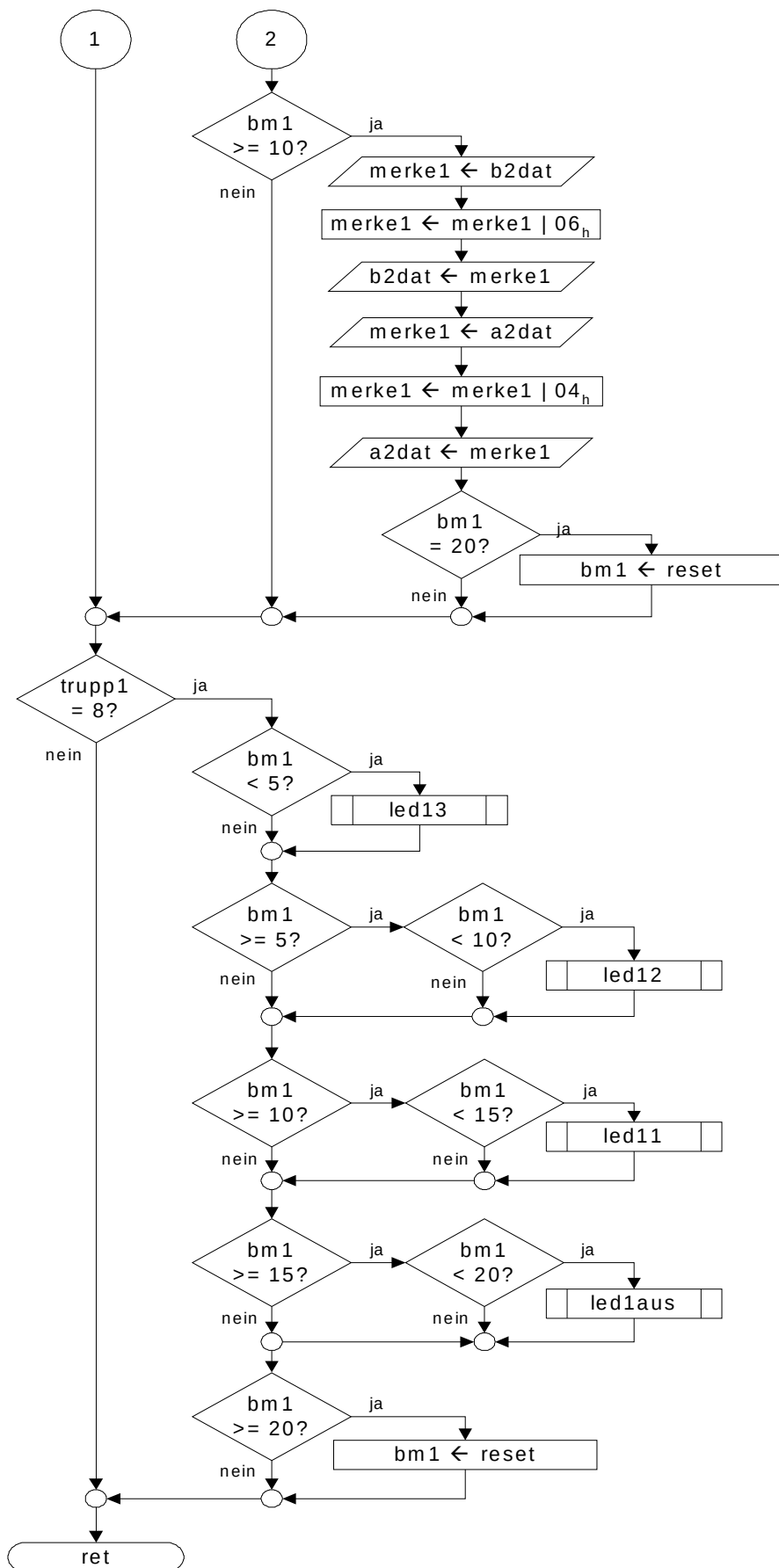
	Port	Maske (Hex)
blink2()	b2dat	30
	a2dat	22
blink3()	b2dat	80
	b1dat	01
	a1dat	01

Beim Wert 8 von ‚trupp1‘ hat eine „Lauflichtfolge nach links“ zu erscheinen. Ähnlich wie zuvor beschrieben, erfolgt nun Kontrollprozedur von ‚bm1‘ mit daraus resultierenden Aktionen:

Wert: bm1 <5	Aktion:	led13()
Wert: bm1 >=5 & <10	Aktion:	led12()
Wert: bm1 >=10 & <15	Aktion:	led12()
Wert: bm1 >=15 & <20	Aktion:	led1aus()
Wert: bm1 >=20	Aktion:	bm1 ← reset

5.7.16.2 Programmablaufplan ,blink1()'





5.8 Datenübermittlung zwischen System und PC

Aufgrund der Zeitknappheit durch die aufgetretenen Probleme, entschieden wir uns, die Entwicklung der Software für die gesamte Datenübertragung vom System zum PC außer Acht zu lassen.

Jedoch haben wir sämtliche Unterprogramme so gestaltet, dass in einer später zu entwickelnden Übertragungsroutine lediglich die Daten übermittelt und nicht zuvor noch aufbereitet werden müssen.

Alle relevanten Daten stehen nach einem Einsatz in der Variable des Typs Vektorvariable ‚Daten[][][]‘ bereit und werden durch den gepufferten RAM-Baustein auf lange Zeit gespeichert.

Der Inhalt ist nach folgendem System abgespeichert:

Daten [W][X][Y][Z] →

W	Einsatznummer, fortlaufend
X	Truppnummer, fortlaufend
Y	Truppmitglied
Z	Einsatzrelevante Daten
0	PID-Nummer
1	Geräteart
2	Druck vor dem Einsatz
3	Druck bei Ankunft der Einsatzstelle
4	Druck bei Ende des Einsatzes
5	verrichtete Tätigkeit

In den vorherigen Modulbeschreibungen wurde diese Vektorvariable auch „Vektorspeicher“ oder „Speicherbereich“ genannt, da hier alle zu „speichernden“ Daten eingetragen werden.

6 Inbetriebnahme

6.1 Testphase

Nachdem die Hardware komplett entwickelt und bestückt war, konnten alle System-Komponenten das erste Mal per Flachbandkabel verbunden und in Betrieb genommen werden. Jedoch geschah bei der ersten Spannungsaufschaltung nicht das Geringste. Demnach haben wir alle zusätzlichen Komponenten vom System abgeklemmt und somit vorerst nur eine absolut minimale Systemkonfiguration mit nur einer PIO und einem Display in Betrieb genommen. Das zugehörige Softwareprogramm war ebenfalls minimalistisch und beinhaltete nur die reine Initialisierung des einen PIO-Controllers. Da wir mit diesem ersten Testprogramm noch keine Displayinitialisierung vorgenommen hatten, aber irgendeine Diagnose des Systembusses benötigten, bedienten wir uns einer vorhandenen Systembus-Diagnoseplatine, die über einen 44-poligen Stecker mit dem Systembus unseres Mikrocontrollersystems verbunden wurde. Die Diagnoseplatine signalisiert mit Hilfe von LED's auf einfachste Weise alle wichtigen Steuersignale sowie den kompletten Adressbus und Datenbus. Um die Adressen, Datenwörter und Steuersignale während der Diagnose überhaupt aussagefähig darstellen zu können, ist es wichtig, dass die CPU nicht mit dem systeminternen Quarz angesteuert wird, sondern über einen manuell generierten Takt von der Diagnoseplatine. Der so genannte „Single-Step-Betrieb“. Dazu ist es jedoch notwendig, den Systemtakt zu unterbrechen. In unserem Fall haben wir diesbezüglich bei der nachträglichen Entwicklung der CPU-Extension-Platine einen Jumper eingebaut, womit man den Systemtakt unterbrechen kann.

Somit waren nun alle Vorbereitungen für einen vernünftigen Testbetrieb getätigt. Folglich konnten wir eindeutige Zustände auf dem Systembus, Datenbus und Steuerbus Schritt für Schritt ablesen und die Befehle sowie die Datenwörter mit unserem Testprogramm ‚check1()‘ in der folgenden Tabelle verifizieren.

6.1.1 Testprotokoll des ersten Testprogramms ,check1()'

	Hex-Code	Dual-Code								Angezeigter Dual-Code							
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	0	
✓	CF	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	02	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	FE	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	02	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	CF	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	03	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✗	F8	1	1	1	1	1	0	0	0	1	1	1	1	1	0	1	0
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	03	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	CF	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	12	0	0	0	1	0	0	1	0	0	0	1	1	0	0	1	0
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	DB	1	1	0	1	1	0	1	1	1	1	0	1	1	0	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	12	0	0	0	1	0	0	1	0	0	0	1	1	0	0	1	0
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	CF	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	13	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✗	49	0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	13	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	FF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	00	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	01	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	10	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✓	11	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✓	FD	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✗	11	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	1
✓	3E	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	0
✗	00	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
✓	D3	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1
✓	11	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Wie man dem obigen Protokoll entnehmen kann, gab es bereits im ersten Testdurchlauf mit minimaler Hardware diverse Zustände auf dem Datenbus, die nicht dem Programmablauf entsprachen. Es traten nach und nach sporadische Fehlerzustände auf, obwohl wir die Hardware als auch die Software in verschiedenster Weise in diversen Varianten änderten, um einzelne Funktionen und Bausteine als Fehlerquelle auszuschließen.

6.2 Aufgetretene Probleme und Lösungen

Ein grundlegendes Problem war bereits das angesprochene Problem mit den Fehlerzuständen auf dem Datenbus, welches uns einiges Kopfzerbrechen bereitete. Zunächst hatten wir sämtliche Hardwarekomponenten auf Kurzschlüsse etwaiger Leiterbahnen sowie falsche oder fehlerhafte Kabelverbindungen überprüft. Nachdem diese Suche erfolglos war, tauschten wir identische Komponenten (Displays) untereinander aus, denn zwischenzeitlich hatten wir festgestellt, dass die Fehlerzustände des Datenbusses nur in Zusammenhang mit den angeschlossenen Displays immer an den gleichen Datenleitungen (D2 und D5) auftraten. Da sich der Fehler systematisch auf den ganzen Systembus auswirkte lag die Vermutung nahe, dass evtl. die CPU mit der Vielzahl von Komponenten am Systembus überlastet wäre und die Ströme des Datenbusses nicht mehr „treiben“ könnte. Jedoch hatten wir bereits ein minimales System mit nur einer PIO und einem Display, also musste der Fehler definitiv mit der Displayplatine zusammenhängen. Konstruktionsfehler der Displayplatine konnten wir nach einer sorgfältigen Prüfung ebenfalls ausschließen. Da wir Displays mit integriertem Controller verwendet haben, wussten wir nicht genau, wie der Datenbus intern vom Hersteller verschaltet wurde. Nach einem Anruf beim Displayhersteller Electronic-Assembly erfuhren wir, dass die Displays intern je einen Pull-Up-Widerstand am Datenbus besitzen. Das stützte wiederum unsere Annahme, dass die Ausgänge der CPU nicht in der Lage wären, die Ströme des Datenbusses zu „treiben“, also z.B. einen Pegel von High auf Low zu „ziehen“, wenn die internen Pull-Up-Widerstände des Displaycontrollers am Datenbus evtl. zu klein wären. Aufgrund dieser Annahme erweiterten wir unser Systembus testweise mit einem Bustreiber, der die Datenbussignale zwischen der CPU und den Displays bidirektional verstärkte. Dabei mussten wir zu jedem Taktzyklus darauf achten, in welche Richtung der Datenfluss verstärkt werden sollte und nutzten das Signal RD (Read) von der CPU, welches signalisiert ob die CPU gerade lesen oder schreiben will, um die Richtung des Bustreibers umzuschalten. Leider hatten wir durch diese Maßnahme nur mäßigen Erfolg, denn wir konnten mit diesem Aufbau nur ein Display ansteuern. Sobald ein zweites oder drittes Display angeschlossen wurde, gab es wieder Kollisionen auf dem Datenbus.

Nach diversen Änderungsversuchen und wiederholter genauer Prüfung sämtlicher Systemkomponenten und Vergleich aller Steckerbelegungen, sowie komplette Signalverfolgung zu allen Bausteinen, stellte sich heraus, dass die Displays immer dann auf den Datenbus zugegriffen haben, als sie eigentlich gesperrt waren. Demnach war das Aktivierungssignal (CE) genau invertiert und folglich konnten wir den Fehler schnell lokalisieren.

Im Endeffekt wurde dieser Fehler durch einen falschen Logik-IC in der Displayvorsaltung, welche, wie in Kapitel DISTAS beschrieben ist, die Verriegelung des Aktivierungssignals (CE) bei falscher Softwareansteuerung übernimmt, verursacht. Der IC3A auf der DISTAS-Platine muss korrekterweise ein 7408 (UND-Gatter) sein. Bei der Entwicklung mit dem Layouteditor EAGLE wurde das Bauteil im Schaltplan auch korrekt entworfen. Aufgrund eines Fehlers in der Bauteilbibliothek von EAGLE wurde dieses Bauteil jedoch im Bestückungsplan für die Layoutansicht als 7400 (NAND-Gatter) falsch bezeichnet. Da der Bestückungsplan als Vorlage für die Platinenfertigung diente, wurde dort ein entsprechender Baustein eingebaut. Nach dem Austausch dieses Logik-IC's gegen den richtigen 7408 funktionierte das gesamte System mit allen angeschlossenen Systemkomponenten und ohne zwischenzeitlich eingebautem Bustreiber. Leider hatte uns diese kleine Verwechslung viel Zeit und Nerven gekostet. Dann ging es aber mit großen Schritten voran, denn nachdem wir alle Bauteile in einem Testprogramm initialisieren konnten, fingen wir an, das eigentliche Hauptprogramm zu entwickeln und die bereits vorentwickelten Routinen zu implementieren.

Bei der Ansteuerung der Displayhintergrundbeleuchtung trat ein weiterer seltsamer Fehler auf, denn bereits bevor die Hintergrundbeleuchtung über die Software angesteuert wurde, flackerte die Displaybeleuchtung sporadisch und wurde heller und dunkler, obwohl sie nur EIN- oder AUS-geschaltet werden kann. Aufgrund des begrenzten Ausgangsstromes der PIO's, mussten wir die Hintergrundbeleuchtung jeweils über einen FET-Transistor (BS170) ansteuern. Wie bereits bekannt war, musste generell jeder Eingang einer PIO einen Pull-Up- oder Pull-Down-Widerstand für ein definiertes Signal besitzen, bei Ausgängen war dies jedoch nicht notwendig. Somit haben wir am Ausgang der PIO für die Hintergrundbeleuchtung keinen Widerstand gesetzt. Dabei haben wir nicht bedacht, dass wir mit dem Ausgang der PIO direkt den FET BS170 ansteuern, der einen Eingang darstellt. Da der FET am Eingang sehr hochohmig ist und somit bereits sehr geringe Ströme ausreichen ihn durchzusteuern, beobachteten wir das Phänomen, dass unsere Anschlussleitung zum FET eine Antennenwirkung zeigte und somit die Helligkeit der Hintergrundbeleuchtung mit dem Überschweben der bloßen Hand, analog zum mehr oder minder durchgesteuerten FET, variierte. Durch das einlöten eines Pull-down-Widerstandes konnten wir diesen Fehler schnell beheben.

Ein weiteres Problem zeigte sich beim Einlesen der Taster mit Hilfe der Interruptroutinen. Wie bereits im Kapitel DISTAS beschrieben, haben wir Taster mit einer bereits sehr geringen Prellzeit von 100µs verwendet und somit vorerst auf eine Hardwareentprellschaltung verzichtet, weil die PIO zum Abfragen des Ports eine höhere Zykluszeit benötigt und somit das Prellen des Tasters im Normalfall nicht registrieren kann. Jedoch traten sporadische Fälle auf, in denen die Interruptroutinen bei einem Tastendruck mehrmals aufgerufen wurden und in eine unvorhergesehene Programmschleife sprangen. Da dieses Problem ebenfalls mit einer Entprellschaltung auftrat, haben wir das geringe Prellen des Tasters hierfür ausgeschlossen. Folglich kamen wir zur Annahme, dass vielmehr der Zeitpunkt der Abfrage des PIO-Ports dafür relevant sei, denn durch das Prellen des Tasters liegt für eine sehr kurze Zeit ein nicht definiertes Datenwort an dem Port der PIO. Wenn die PIO exakt zu diesem Zeitpunkt gerade das maskierte Datenwort einliest, kann dies zu undefinierten Zuständen oder zum Mehrfachauslösen der Interruptroutine führen. Um dieses auf einfachste Weise zu verhindern, haben wir jeweils einen definierten Zeitraum zwischen 50ms und 100ms vor und nach dem Bearbeiten der Interruptroutine als Wartezeit programmiert. Somit wartet das System nach einem Tastendruck diesen Zeitraum ab, so dass der Zustand am entsprechenden PIO-Port nach der Prellzeit eindeutig ist und erst dann die Interruptroutine ausgeführt wird. Durch diese Methode konnten wir dieses Problem einfach und zuverlässig umgehen.

7 Schlussbetrachtung

Abschließend kann man die Arbeit an diesem Projekt als „Berg-und-Tal-Fahrt“ bezeichnen. Aufgrund einiger Probleme, in Hard- sowie Software, deren Lösungsfindung mehr Zeit in Anspruch nahm als erwartet, mussten wir feststellen, dass die uns zur Verfügung stehende Zeit für unser Vorhaben nicht ausreichend war. Somit waren wir gezwungen ein paar kleinere Veränderungen, in Form von Einsparungen vorzunehmen. Wir entschieden uns, die Ladeelektronik, die Akkuzustandsanzeige und die serielle Datenübertragung softwareseitig innerhalb dieses Zeitraumes nicht zu verwirklichen und richteten unser Hauptaugenmerk auf die eigentliche Funktion des Gerätes. Dennoch ließen wir uns zu keiner Zeit entmutigen, denn jedes noch so kleine Erfolgserlebnis gab uns neuen Ansporn.

Viele Probleme wurden in Eigeninitiative gelöst, dennoch sind Probleme aufgetreten, bei deren Lösung wir nicht weiter kamen und somit die projektbetreuenden Lehrer um Rat baten. Im Großen und Ganzen war die Arbeit an dem Projekt sehr interessant und lehrreich. Die Arbeit im Team verlief in der Regel reibungslos, obwohl man an einigen Diskussionen einfach nicht vorbei kam. Des Weiteren haben sich die verschiedenen Teams auch untereinander geholfen, was die gesamte Projektarbeit positiv beeinflusst hat.

8 Projekttagbuch

1. Woche	- Gruppenaufteilung (Software / Hardware) - Problemdiskussion über Hard- und Softwarerealisierungen - Ermittlung der Steuerwörter sowie Basisadressen für sämtliche PIOs - Erstellung der ersten Materialliste - Erstellung eines Blockschaltbildes der vorläufigen Lösung
2. Woche	- Erstellung der ersten Einkaufsliste (Dauer ca. 2 Stunden!) - Informationen (Eigenschaften sowie Belegung) über RAM und ROM gesammelt - erste PID-IN-Platinen geätzt, leider Fehlerhaft, da Lötunkte zu klein - Diverse Bibliotheken in Eagle optimiert - Bearbeitung des Layouts der PIO-Platine
3. Woche	- Bearbeitung der Platinen-Layouts (PIO-Platine, PIDS, DISTAS) - Softwareentwicklung (Erstellen von Testprogrammen) - erneutes Ätzen und Bohren der PID-IN-Platinen - Layout für beidseitige PIO-Platine fertig gestellt - PIO-Platine geätzt, festgestellt, dass es mit Schulmitteln zu riskant ist
4. Woche	- PID-IN-Platinen bestückt - Layout der PIO-Platine geändert - Entprellschaltung entwickelt - Zusatzschaltung für Displays entwickelt, um Kollisionen auszuschließen - Taster mit Oszilloskop getestet, festgestellt, dass diese teilweise entprellt sind - PIO-Platinenlayout zum ätzen geschickt
5. Woche	- DISTAS-Platine entwickelt - DISTAS-Platine geätzt und gebohrt - PIDS geätzt und gebohrt - Einarbeitung in die Übertragung per serieller Schnittstelle (SIO) in JAVA
6. Woche	- PIO-Platine eingetroffen, durchmessen der Verbindungen und Leiterbahnen - DISTAS-Platinen geätzt und gebohrt - Bestückung der PIO-Platine und DISTAS-Platinen - JAVA-Programmierung für SIO - Berechnung von Vor-, Pull-Up- und Pull-Down-Widerständen
7. Woche	- Adressierung neu erstellt - Vorläufige Restarbeiten der Hardware (Kabel gefertigt, Taster angelötet etc.) - Vorbereitung für ersten Hardwaretest - Erstellung eines Assemblerprogramms zum Testen der Hardware

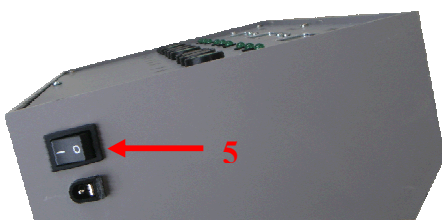
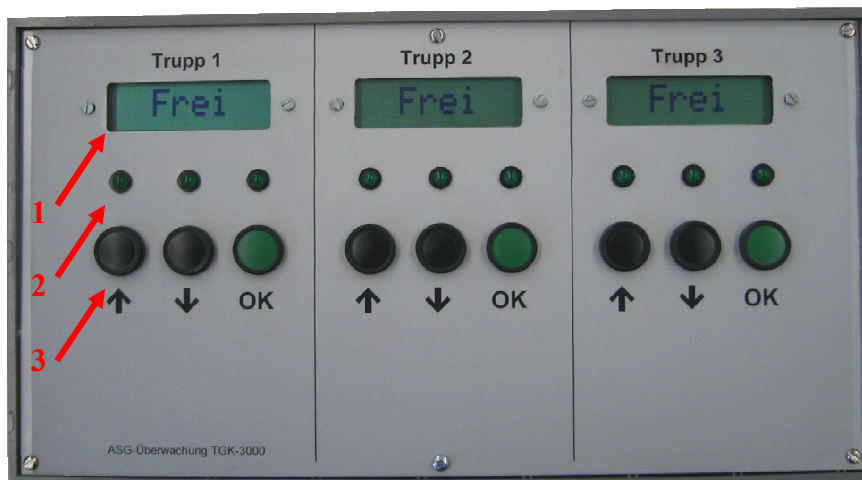
8. Woche	- Erster Hardwaretest - Weitere Hardwaretests folgen - Erste Fehler auf dem Datenbus treten auf - Fehlersuche
9. Woche	- Fehlersuche
10. Woche	- Fehlersuche - Am Ende der Woche Fehler in der Hardware gefunden (Displayvorschaltung)
11. Woche	- Weitere Softwaretests folgen - Erste Darstellung von Zeichen auf allen drei Displays - Einlesen der PIDS - Erste Softwareprobleme treten auf
12. Woche	- Weitere Softwareentwicklung - Softwareprobleme werden gelöst, neue entstehen - Beginn der Dokumentation
13. Woche	- Softwareentwicklung - Gehäuseeinbau vorbereiten (Vermessen, Zeichnen, Bohren, Fräsen) - Dokumentation

9 Bedienungsanleitung TGK3000

9.1 Allgemeines

Die TGK3000 ist eine mobile Einheit zum Überwachen sämtlicher Feuerwehrtätigkeiten (ob Einsatz oder Übung), in denen Atemschutzgeräte verwendet werden. Um die ohnehin enorme Stresssituation im Feuerwehreinsatz für diesen Teil zu entspannen, bietet dieses Gerät eine einfache Menüführung. Je Trupp dienen drei Taster zum Steuern während der gesamten Überwachung. Es ist eine Überwachung von bis zu 3 unabhängig eingesetzten Trupps möglich. Auch bei Dunkelheit ist die TGK3000 durch die Hintergrundbeleuchtung der Displays einsetzbar.

9.2 Aufbau der der TGK3000



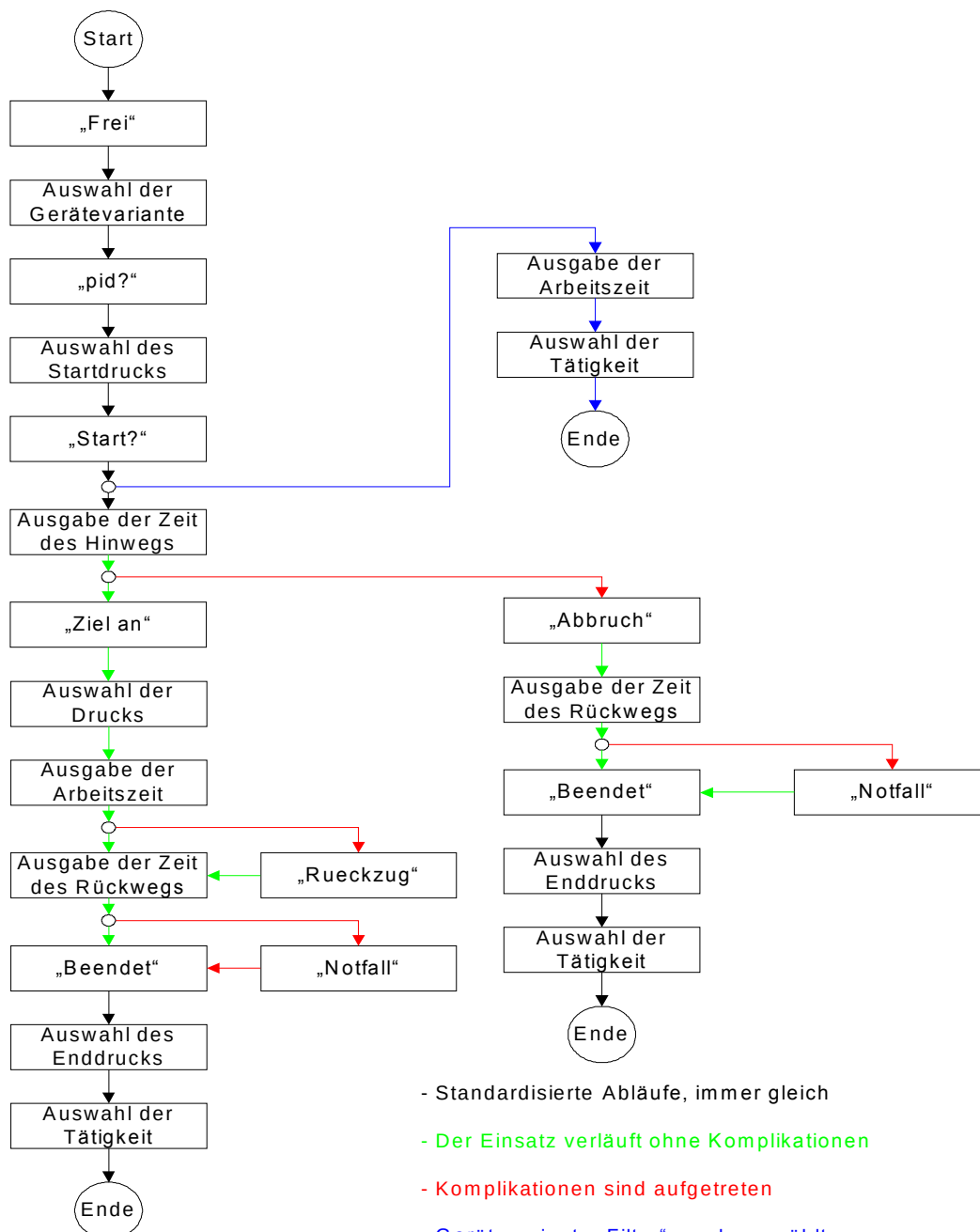
Erläuterung der einzelnen Elemente:

- 1 → Display, je Trupp
 - 2 → Status- LED, 3 je Trupp
 - 3 → Taster, 3 je Trupp
 - 4 → Lichttaster, Truppübergreifend
 - 5 → Systemhauptschalter
- An der Vorderseite des Geräts befinden sich die Buchsen für die PID- Sticks

9.3 Menüstruktur der TKG3000

Eine klar strukturierte und einfach verständliche Menüführung begleitet den Überwachenden für die gesamte Dauer des Einsatzes. Die Bedienung der einzelnen Truppüberwachungsmodule erfolgt unabhängig und geschieht über separate Bedienelemente.

Die Softwarestruktur lässt sich wie folgt visualisiert darstellen:



Einschalten

Schaltet man das Gerät am Hauptschalter ein, so wird zunächst ein kompletter Selbsttest des Geräts durchgeführt. Alle 3 Displays, sowie die 9 LED's leuchten auf und der akustische Signalgeber ertönt. Nach 1s werden all diese Elemente ausgeschaltet, der Selbsttest ist abgeschlossen und die einzelnen Truppmodule befinden sich im Vorbereitungsmodus. Nun werden alle 3 Module durch die Displays als FREI und verfügbar (Abb. 1) signalisiert. Von diesem Punkt an erfolgt die Bedienung über die jeweiligen 3, dem Trupp zugehörigen, Taster.



Gerätetypwahl

Zunächst wird über Taste ‚OK‘ die Datenaufnahme für den Trupp begonnen. Im Display erscheint der erste zur Verfügung stehende Gerätetyp (Abb. 2). Über die Auswahl Tasten (Pfeiltasten) ist es nun möglich, die tatsächlich verwendete Gerätevariante zu wählen. Die Bestätigung der Auswahl erfolgt schließlich über ‚OK‘ und wird durch ein akustisches Signal quittiert.



PID-Stick einlesen

Nun wird der Bediener des Gerätes aufgefordert die PID-Sticks der Truppmitglieder in die PID-Buchsen einzusetzen. Ist dies erfolgt, oder bereits zuvor geschehen, so wird der Menüpunkt akustisch quittiert und ohne weitere Bestätigungen verlassen.

Eingeben der Behälterdrücke

Als nächstes folgt die Eingabe der Behälterdrücke vor dem Einsatz. Dies geschieht für die einzelnen Truppmitglieder nacheinander. Durch die LED's 1 bis 3 wird angezeigt, von welchem Truppmitglied der Druck abgefragt wird. Eingestellt werden die Drücke mittels der Pfeiltasten und anschließend mit ‚OK‘ ausgewählt. Sind für alle Truppmitglieder die Werte gespeichert erfolgt die akustische Bestätigung und die Vorbereitungsphase ist beendet.



Einsatzmodus

Der vorbereitete Trupp ist nun vollständig Einsatzbereit. Dies wird im Display durch ‚Start‘ signalisiert.



Einsatzbeginn

Beginnt schließlich der Einsatz so ist nur die Bestätigung durch die Taste ‚OK‘ erforderlich und die Einsatzzeit für den Hinweg wird vom Maximalwert (ergibt sich durch die Gerätetypauswahl) heruntergezählt, im Display angezeigt und im Minutentakt aktualisiert. Der Einsatzstatus ‚Hinweg‘ ist durch ein großes ‚H‘ im Display erkennbar.



Meldung ‚Einsatzort an‘

Beim Eintreffen des Trupps an der eigentlichen Einsatzstelle meldet sich dieser über sein mitgeführtes Handsprechfunkgerät und gibt die Behälterdrücke der einzelnen Mitglieder durch. Diese werden nun vom Überwachenden, wie zuvor, eingegeben und bestätigt.



Arbeiten an der Einsatzstelle

Automatisch wird anschließend die noch verbleibende Zeit zum Arbeiten an der Einsatzstelle (vom System berechnet) im Display angezeigt und im Minutentakt aktualisiert. Ähnlich wie bei der Zeit auf dem Weg wird auch die Zeit des Arbeitens durch ein großes ‚A‘ im Display kenntlich gemacht.



Ende der Arbeitszeit ohne vorherigen Rückzugsantritt

Ist die errechnete Arbeitszeit abgelaufen ohne das zuvor weitere Eingriffe durch den Bediener am Modul durchgeführt wurden, ertönt ein akustisches Dauerwarnsignal und im Display erscheint die Aufforderung ‚Rueckzug‘. Das Warnsignal sowie die Rückzugsanordnung werden durch die Taste ‚OK‘ beendet, wenn der Überwachende dem Trupp den Rückzugsbefehl gegeben hat und der Rückzugsmodus wird erreicht.

Rückzugsantritt vor Ende der Arbeitszeit

Ist vor Ablauf der Arbeitszeit der Rückzug angetreten worden, dies geschieht über die ‚OK‘ mit folgender akustischer Signalisierung, bleibt die akustische und optische Warnung aus und auch hier wird der Rückzugsmodus erreicht.

Rückzugszeit

In diesem Modus, gekennzeichnet durch ein großes ‚R‘ im Display, wird die Zeit von 1 an aufwärts gezählt, damit der Überwachende immer im Blick hat, wie lange sich der Trupp bereits auf dem Rückweg befindet. Meldet sich dieser nun bei der Atemschutzüberwachung vollzählig zurück, so wird der Einsatz mit der Taste ‚OK‘ beendet.



Behälterdruck bei Einsatzende

Im Anschluss an den Einsatz sind für die spätere Einsatzdokumentation zunächst noch die Behälterdrücke der einzelnen Truppmitglieder, wie zuvor, einzugeben.

Angabe der durchgeführten Tätigkeit

Als letzte Eingabe für den soeben beendeten Einsatz ist die zuvor durchgeführte Tätigkeit aufzunehmen. Die Auswahl erfolgt erneut über die beiden Pfeiltasten. Durch Betätigen der von ‚OK‘ wird die durchgeführte Arbeit in den Speicher übernommen und der Einsatz ist beendet.

Im Anschluss wird wieder der Betriebszustand ‚Vorbereitung‘ erreicht und der nächste Trupp kann für seinen Einsatz vorbereitet werden.

Sonderfall: Dauer für den Hinweg macht Arbeiten unmöglich

Tritt der Fall ein, dass der Hinweg so lange dauert bis eine gesicherte Rückzugszeit nicht mehr gewährleistet werden kann, so ertönt ein akustisches Warnsignal und im Display erscheint ‚Abbruch!‘ Diese Signalisierung wird durch ‚OK‘ zurückgesetzt und das Programm läuft im ‚Rückzug‘-Modus, jedoch mit der Signalisierung ‚RA‘ im Display, weiter.

9.4 Auswahlmöglichkeiten der einzelnen Abfragen

Geräteart: PA 300, ChemieSA, Hitzesch, Regen PA, Filter

Tätigkeit: Retten, Loeschen, Erkunden, Dichten, Auffange, Umpumpen, Funkverb

10 Tabellen

10.1 Signallisten

Signal	CPU	ST16	RAM	ROM	PIO	SIO	ST6 = ST14	ST8a/b	ST9a/b	ST10a/b	ST05	ST01
A0	30	25	10	10	6	-	9	5	5	5	-	-
A1	31	23	9	9	5	12	10	4	4	4	-	-
A2	32	21	8	8	-	-	17	-	-	-	-	-
A3	33	19	7	7	-	-	-	-	-	-	-	-
A4	34	17	6	6	-	-	-	-	-	-	-	-
A5	35	15	5	5	-	-	-	-	-	-	-	-
A6	36	13	4	4	-	-	18	-	-	-	-	-
A7	37	11	3	3	-	-	19	-	-	-	-	-
A8	38	9	25	25	-	-	-	-	-	-	-	-
A9	39	7	24	24	-	-	-	-	-	-	-	-
A10	40	5	21	21	-	-	-	-	-	-	-	-
A11	1	40	23	23	-	-	-	-	-	-	-	-
A12	2	14	2	2	-	-	-	-	-	-	-	-
A13	3	6	26	26	-	-	-	-	-	-	-	-
A14	4	8	1	27	-	-	-	-	-	-	-	-
A15	5	10	-	-	-	-	-	-	-	-	-	-
D0	14	26	11	11	19	27	1	7	7	7	-	-
D1	15	34	12	12	20	28	2	8	8	8	-	-
D2	12	22	13	13	1	1	3	9	9	9	-	-
D3	8	35	15	15	40	2	4	10	10	10	-	-
D4	7	12	16	16	39	5	5	11	11	11	-	-
D5	9	16	17	17	38	6	6	12	12	12	-	-
D6	10	18	18	18	3	7	7	13	13	13	-	-
D7	13	24	19	19	2	8	8	14	14	14	-	-
CLK	6	20	-	-	25	20	14	-	-	-	-	-
5V	11	42/44	28	28	26	-	-	2	2	2	-	-
/INT	16	36	-	-	23	-	15	-	-	-	-	-
/NMI	17	32	-	-	-	-	-	-	-	-	-	-
/HALT	18	-	-	-	-	-	-	-	-	-	-	-
/MREQ	19	37	-	-	-	-	-	-	-	-	-	-
/IORQ	20	30	-	-	36	-	12	-	-	-	-	-
GND	29	28/27	-	-	11	-	-	1	1	1	-	-
/RFSH	28	-	-	-	-	-	-	-	-	-	-	-
/M1	27	31	-	-	37	-	11	21	21	21	-	-
/RES	26	33	-	-	-	21	-	-	-	-	-	-
/BUSRQ	25	-	-	-	-	-	-	-	-	-	-	-
/WAIT	24	38	-	-	-	-	-	-	-	-	-	-
/BUSACK	23	-	-	-	-	-	-	-	-	-	-	-
/WR	22	39	27	27	-	10	-	-	-	-	-	-
/RD	21	43	22	22	35	13	13	23	23	23	-	-
Daisy-Chain	-	-	-	-	(IEI:24 / IEO:22)	-	16	-	-	-	-	-

Atemschutzgeräteüberwachung

Signal	CPU	ST16	RAM	ROM	PIO	SIO	ST6 = ST14	ST8a/b	ST9a/b	ST10a/b	ST05	ST01
2Y0	-	-	-	-	4	-	-	-	-	-	-	-
2Y1	-	-	-	-	4	-	-	-	-	-	-	-
2Y2	-	-	-	-	-	11	-	-	-	-	-	-
2Y3	-	-	-	-	-	-	20	-	-	-	-	-
3Y0	-	-	-	-	4	-	-	-	-	-	-	-
3Y1	-	-	-	-	4	-	-	-	-	-	-	-
3Y2	-	-	-	-	4	-	-	-	-	-	-	-
3Y3	-	-	-	-	4	-	-	-	-	-	-	-
3Y4	-	-	-	-	4	-	-	-	-	-	-	-
3Y5	-	-	-	-	-	-	-	6	-	-	-	-
3Y6	-	-	-	-	-	-	-	-	6	-	-	-
3Y7	-	-	-	-	-	-	-	-	-	6	-	-
1T1	-	-	-	-	-	-	-	3	-	-	1	2c
1T2	-	-	-	-	-	-	-	15	-	-	2	2a
1T3	-	-	-	-	-	-	-	16	-	-	3	3c
1LED1	-	-	-	-	-	-	-	18	-	-	4	3a
1LED2	-	-	-	-	-	-	-	19	-	-	5	4c
1LED3	-	-	-	-	-	-	-	20	-	-	6	4a
2T1	-	-	-	-	-	-	-	-	3	-	7	5c
2T2	-	-	-	-	-	-	-	-	15	-	8	5a
2T3	-	-	-	-	-	-	-	-	16	-	9	6c
2LED1	-	-	-	-	-	-	-	-	18	-	10	6a
2LED2	-	-	-	-	-	-	-	-	19	-	11	7c
2LED3	-	-	-	-	-	-	-	-	20	-	12	7a
3T1	-	-	-	-	-	-	-	-	-	3	13	8c
3T2	-	-	-	-	-	-	-	-	-	15	14	8a
3T3	-	-	-	-	-	-	-	-	-	16	15	9c
3LED1	-	-	-	-	-	-	-	-	-	18	16	9a
3LED2	-	-	-	-	-	-	-	-	-	19	17	19a
3LED3	-	-	-	-	-	-	-	-	-	20	18	19c

10.2 Steckerbelegungen

ST01		
PIN	A	C
1	+5V	+5V
2	IC4-B0	IC4-A0
3	IC4-B1	IC4-A1
4	IC4-B2	IC4-A2
5	IC4-B3	IC4-A3
6	IC4-B4	IC4-A4
7	IC4-B5	IC4-A5
8	IC4-B6	IC4-A6
9	IC4-B7	IC4-A7
10	IC4-ASTB	-
11	IC4-ARDY	IC4-BRDY
12	-	-
13	+10V	IC4-BSTD
14	-	NMI
15	-	RES-out
16	-	-
17	IC3-BRDY	IC3-ARDY
18	IC3-BSTD	IC3-ASTD
19	IC3-A0	IC3-B0
20	IC3-A1	IC3-B1
21	-	-
22	-10V	-
23	-	-
24	IC3-A2	IC3-B2
25	IC3-A3	IC3-B3
26	IC3-A4	IC3-B4
27	IC3-A5	IC3-B5
28	IC3-A6	IC3-B6
29	IC3-A7	IC3-B7
30	-	RES-in
31	-	-
32	GND	GND

ST02	
1	ARDY
2	BRDY
3	ASTD
4	BSTD
5	B7
6	A7
7	B6
8	A6
9	B5
10	A5
11	B4
12	A4
13	B3
14	A3
15	B2
16	A2
17	B1
18	A1
19	B0
20	A0

ST03	
1	-10V
2	Masse
3	+10V
4	NC
5	DSR-V24
6	DSR
7	CTS
8	CTS-V24
9	RXD
10	RXD-V24
11	DTR
12	DTR-V24
13	RTS-V24
14	RTS
15	TXD
16	TXD-V24
17	RXC
18	TXC
19	Sync
20	NC

ST04	
1	+5V
2	GND
3	+5V
4	GND
5	+5V
6	GND

Atemschutzgeräteüberwachung

ST05	
1	1T1
2	1T2
3	1T3
4	1LED1
5	1LED2
6	1LED3
7	2T1
8	2T2
9	2T3
10	2LED1
11	2LED2
12	2LED3
13	3T1
14	3T2
15	3T3
16	3LED1
17	3LED2
18	3LED3
19	LCD-Bel.
20	-

ST06	
1	D0
2	D1
3	D2
4	D3
5	D4
6	D5
7	D6
8	D7
9	A0 (/A B)
10	A1 (C D)
11	/M1
12	/IOREQ
13	/RD
14	CLK
15	/INT
16	Daisychain
17	A2
18	A6
19	A7
20	2Y3

ST07	
1	+5V
2	+5V
3	+5V
4	+5V
5	-
6	-
7	GND
8	GND
9	GND
10	GND

ST08 A (B)	
1	GND
2	+5V
3	1T1
4	A1 (C D)
5	A0 (R W)
6	2Y5 (CE)
7	D0
8	D1
9	D2
10	D3
11	D4
12	D5
13	D6
14	D7
15	1T2
16	1T3
17	1LED1
18	1LED2
19	1LED3
20	M1
21	-
22	RD
23	-
24	-
25	-
26	-

ST09 A (B)	
1	GND
2	+5V
3	2T1
4	A1 (C D)
5	A0 (R W)
6	2Y6 (CE)
7	D0
8	D1
9	D2
10	D3
11	D4
12	D5
13	D6
14	D7
15	2T2
16	2T3
17	2LED1
18	2LED2
19	2LED3
20	M1
21	-
22	RD
23	-
24	-
25	-
26	-

ST10 A (B)	
1	GND
2	+5V
3	3T1
4	A1 (C D)
5	A0 (R W)
6	2Y7 (CE)
7	D0
8	D1
9	D2
10	D3
11	D4
12	D5
13	D6
14	D7
15	3T2
16	3T3
17	3LED1
18	3LED2
19	3LED3
20	M1
21	-
22	RD
23	-
24	-
25	-
26	-

ST11 A (B)	
1	1A0
2	1A1
3	1A2
4	1A3
5	1A4
6	1A4
7	1A6
8	GND
9	2B0
10	2B1
11	2B2
12	2B3
13	2B4
14	2B5
15	2B6
16	GND
17	3A0
18	3A1
19	3A2
20	3A3
21	3A4
22	3A5
23	3A6
24	GND
25	-
26	-

ST12 A (B)	
1	4B0
2	4B1
3	4B2
4	4B3
5	4B4
6	4B5
7	4B6
8	GND
9	5A0
10	5A1
11	5A2
12	5A3
13	5A4
14	5A5
15	5A6
16	GND
17	6B0
18	6B1
19	6B2
20	6B3
21	6B4
22	6B5
23	6B6
24	GND
25	-
26	-

ST13 A (B)	
1	7A0
2	7A1
3	7A2
4	7A3
5	7A4
6	7A4
7	7A6
8	GND
9	8B0
10	8B1
11	8B2
12	8B3
13	8B4
14	8B5
15	8B6
16	GND
17	9A0
18	9A1
19	9A2
20	9A3
21	9A4
22	9A5
23	9A6
24	GND
25	-
26	-

ST16	
1	-
2	-
3	-
4	-
5	A10
6	A13
7	A9
8	A14
9	A8
10	A15
11	A7
12	D4
13	A6
14	A12
15	A5
16	D5
17	A4
18	D6
19	A3
20	CLK
21	A2
22	D2
23	A1
24	D7
25	A0
26	D0
27	GND
28	GND
29	IEO
30	IOREQ
31	M1
32	NMI
33	RESET
34	D1
35	D3
36	INT
37	MREQ
38	WAIT
39	WR
40	A11
41	-
42	Vcc
43	RD
44	Vcc

ST14	
1	D0
2	D1
3	D2
4	D3
5	D4
6	D5
7	D6
8	D7
9	A0
10	A1
11	M1
12	IOREQ
13	RD
14	CLK
15	INT
16	IEO
17	A2
18	A6
19	A7
20	2Y3
26	-

ST15	
1	Daisyhain
2	2Y3
3	-
4	-
5	-
6	-
7	-
8	-
9	-
10	-

10.3 Stücklisten

10.3.1 Z80-EMUF

Name	Bauteil	Anzahl
C 01 – C02	Tantal 10µ/16V	2
CB 01	Kerko 100n	1
CB 02	Tantal 10µ/25V	1
CB 03 – CB 07	Kerko 100n	5
CB 08 – CB 09	Tantal 10µ/25V	2
IC 01	EPROM	1
IC 02	SRAM 8k	1
IC 03 – IC 04	Z80-PIO	2
IC 06	TL7705	1
IC 07	74HC139	1
IC 08	Quarzosz. 2,4576Mhz	1
IC 09	8251	1
IC 10	74HC393	1
IC 11	75189	1
IC 12	75188	1
J 01	2x6pol. Stiftleiste	1
J 02	2x2pol. Stiftleiste	1
J 03	2x8pol. Stiftleiste	1
J 04 – J 05	2x2pol. Stiftleiste	2
R 01 – R 03	Widerstand 4,7 kΩ	3
R 04	Widerstand 680 Ω	1
R 05 – R 07	Widerstand 4,7 kΩ	3
ST 01A	64pol. Stiftleiste	1
ST 01C	VG a c gewinkelt	1
ST 02 – ST 03	2x10pol. Stiftleiste	2
ST 04	2x3pol. Stiftleiste	1

10.3.2 DISTAS

Name	Bauteil	Anzahl
ST 08B	Wannenstecker 26pol.	1
LCD-DIP081	Dotmatrix-Display	1
IC 2	74HC02 (4-fach NOR-Gate)	1
IC 3	74HC08 (4-fach AND-Gate)	1
LED 1 – LED 2	LED 5mm (grün)	2
LED 3	LED 5mm (rot)	1
S 1 – S 3	Taster (Schließer)	3
Q 1	Transistor BS170	1
R 1	Widerstand 10 Ω	1
R 2 – R 4	Widerstand 1,5 k Ω	3
R 5 – R 7	Widerstand 4,7 k Ω	3

10.3.3 PIO-Verteiler-Platine

Name	Bauteil	Anzahl
ST 08A – ST 13A	Wannenstecker 26pol.	6
ST 05 – ST 06	Wannenstecker 20pol.	2
ST 07	Wannenstecker 10 pol.	1
PIO 3 – PIO 7	Z80-PIO	5
MUX 3	Adressdecoder 74HC138	1
L4940V5	Spannungsregler L4940V5	1
K1RUP – K9RUP	8-fach Parallel-Widerstand 4,7 k Ω	9
R1UP – R5RUP	Widerstand 4,7 k Ω	5
G2RUP	Widerstand 4,7 k Ω	1
BR 1	Widerstand 90 Ω	1
T 1	Taster (Schließer)	1
Q 1	Transistor BS170	1
BUZZER	Miniatur Piezo-Summer	1

10.3.4 CPU-Extension-Platine

Name	Bauteil	Anzahl
ST 14	Wannenstecker 20pol.	1
ST 15	Wannenstecker 10pol.	1
ST 16	Wannenstecker 44pol.	1
Z80-CPU	Z80-CPU	1

10.3.5 PID-IN-Platine

Name	Bauteil	Anzahl
J 1 – J 3	8-fach Modularstrecker (RJ45)	3
ST 11B	Wannenstecker 26pol.	1

10.3.6 PID-Stick

Name	Bauteil	Anzahl
S 1	Mini-Dip-Schalter	1
TEKO-10006	Gehäuse	1

10.4 Kostenaufstellung

Baugruppe	Anzahl	Einzelpreis	Gesamtpreis
Kanis-EMUF	1	110,37 €	110,37 €
PIO-Platine	1	69,22 €	71,47 €
PID-Sticks	9	2,15 €	19,35 €
PID-IN	3	1,12 €	3,36 €
DISTAS	3	22,25 €	66,75 €
CPU-Extension	1	1,54 €	1,54 €
Gehäuse	1	17,70 €	17,70 €
Kabel & Stecker	div.	8,61 €	8,61 €
Platinen etc.	div.	34,76 €	34,76 €
Total			333,91 €

11 Erklärung der Projektteilnehmer

Hiermit erklären wir, dass wir das Projekt „Atemschutzgeräteüberwachung“ in der Zeit von Februar 2006 bis Mai 2006 eigenständig geplant und entwickelt haben. Ansprechpartner waren in dieser Zeit lediglich die projektbetreuenden Lehrer.

(Eike Grünhagen)

(Carsten Töben)

(Esmail Kiapur)

12 Quellennachweise

Internet:

- **www.elektronik-kompodium.de**
- **www.kanis.de**
- **www.computerlexikon.com**
- **www.reichelt.de**
- **www.conrad.de**

Literatur:

- **Technical Manual, „Z80 Microprocessor Family – 4th Edition“ , SGS**
- **Helmut Müller/ Lothar Walz, „Mikroprozessortechnik“, Vogel Fachbuchverlag**
- **Ulla Kirch-Prinz/ Peter Prinz, „C für PCs“, IWT-Verlag**
- **Ulla Kirch Prinz/ Peter Prinz, „C – Einführung und professionelle Anwendung“, mitp-Verlag**
- **Unterrichtskolleg**

13 Anhang

13.1 Quellcode

13 Anhang

13.2 Datenblätter